

**Damage Detection and Identification of Finite Element
Models Using State-Space Based Signal Processing**

**A Summation of work completed at the
Lawrence Livermore National Laboratory**

February 1999 to April 2000

G.C. Burnett

April 28, 2000

U.S. Department of Energy

Lawrence
Livermore
National
Laboratory

DISCLAIMER

This document was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the University of California nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or the University of California, and shall not be used for advertising or product endorsement purposes.

This work was performed under the auspices of the U. S. Department of Energy by the University of California, Lawrence Livermore National Laboratory under Contract No. W-7405-Eng-48.

This report has been reproduced
directly from the best available copy.

Available to DOE and DOE contractors from the
Office of Scientific and Technical Information
P.O. Box 62, Oak Ridge, TN 37831
Prices available from (423) 576-8401
<http://apollo.osti.gov/bridge/>

Available to the public from the
National Technical Information Service
U.S. Department of Commerce
5285 Port Royal Rd.,
Springfield, VA 22161
<http://www.ntis.gov/>

OR

Lawrence Livermore National Laboratory
Technical Information Department's Digital Library
<http://www.llnl.gov/tid/Library.html>

Damage detection and identification of Finite Element Models using state-space based signal processing

A summation of work completed at the
Lawrence Livermore National Laboratory

February 1999 to April 2000

by

Gregory C. Burnett, PhD

EE-DSED

April 28th, 2000

Damage detection and identification of Finite Element Models using state-space based signal processing

Abstract

Until recently, attempts to update Finite Element Models (FEM) of large structures based upon recording structural motions were mostly ad hoc, requiring a large amount of engineering experience and skill. Studies have been undertaken at LLNL to use state-space based signal processing techniques to locate the existence and type of model mismatches common in FEM. Two different methods (Gauss-Newton gradient search and extended Kalman filter) have been explored, and the progress made in each type of algorithm as well as the results from several simulated and one actual building model will be discussed. The algorithms will be examined in detail, and the computer programs written to implement the algorithms will be documented.

Table of Contents

Title page	
Abstract	
Table of Contents	
List of Tables	
List of Figures	
	Pages
1. Introduction to the problem and the algorithms	1-9
1.1. The Gauss-Newton gradient search	
1.2. The Extended Kalman Filter	
2. The models	10-23
2.1. Building K given the identification parameters	
2.2. Simulated 5 story structure (5, 10, 50, 400 DOF)	
2.3. Nevada Test Structure (5 DOF)	
2.3.1. Description of the NTS experiment and files	
3. GN results	24-37
3.1. Simulated models	
3.2. NTS 5 DOF model	
4. EKF results	38-55
4.1. Simulated models	
4.2. NTS 5 DOF model	
4.3. Speeding it up	
5. Conclusions	56-57
5.1. Why the 50 DOF model doesn't work	
5.2. Why the NTS 5 DOF model doesn't work	
6. Recommendations for future work	58-59
6.1. How to get started with the EKF algorithms	
7. Appendices	
7.1. Timeline	60-66
7.2. File locations	67

7.3.	References	68
7.4.	Alphabetical listing of filenames and descriptions	69-74
7.5.	Memo "System ID algorithm" of 3/9/99	75-81
7.6.	Memo "Flaw detection and identification algorithm (5 DOF) of 3/15/99	82-94
7.7.	Memo "Integration Filter and Results" of 3/29/99	95-103
7.8.	Draft of paper on the Gauss-Newton algorithm and its strengths and weaknesses for G. Clark. March-April 1999.	104-111
7.9.	Memo "Continuous to discrete transformations" of 6/2/99	112-117
7.10.	Memo "400 DOF progress" of 7/7/99	118-121
7.11.	Memo on "The State of the Art in Vibration-based Structural Damage Identification" tutorial of 9/17/99	122-123
7.12.	Memo "The 5 DOF NTS model using the damped Gauss-Newton algorithm" of 10/18/99	124-132
7.13.	Memo "10 and 50 DOF results with reduced measurements" using the GN algorithm, 10/18/99	133-144
7.14.	Memo "Augmented state vector continuous discrete extended Kalman filter system identification approach", first look at EKF, 12/22/99	145-152
7.15.	Memo "The dependence of convergence time for the EKF on the various noise parameters", 1-D study, 1/4/00	153-161
7.16.	Memo "The performance of the EKF algorithm compared to the Gauss-Newton for the 10 DOF system", 1/25/00	162-165
7.17.	Memo "Estimating model order using the SVD of the generalized Hankel matrix", 2/29/00	166-170
7.18.	Memo "Extended Kalman filter results for 10 DOF, 50 DOF, and NTS 5 DOF", 3/1/00	171-179
7.19.	Memo "Suggestions and comments from advisory meeting", 3/1/00	180-181
7.20.	Memo "Action on suggestions and comments from advisory meeting of 3/1/00", 4/19/00	182-206

List of Tables

- Table 1.1. Relative qualities of the Gauss-Newton and the Extended Kalman Filter algorithms
Page 8.
- Table 2.1. Data for the accelerometers from the NTS experiments. Page 19.
- Table 2.2. Filenames and maximum voltage limits for the NTS experiments. Page 20.
- Table 4.1. The estimation errors of the original estimate and the EKF algorithm results for a single measurement and 500 samples. The errors in bold are the errors for the elements next to the node being observed, the ones in blue are the lowest errors for that particular case and the errors in red are the largest. Note that there is no observed correlation between an observation and the lowest error. Page 39.
- Table 4.2. The results from the 10 DOF experiment where all possible K_{ij} values are used as the parameters. Only the highlighted parameters converged. Page 49.
- Table 4.3. Data concerning the use of an adaptive (2-5) and constant (6) method to restrict the number of times P is calculated. Page 46.
- Table 4.4. Convergence times and identification errors for the same input/output sequences and different DP thresholds and conditions. Every n th with stiffness means that the P_{dot} values associated with the stiffness states were calculated at each time sample and the entire P_{dot} calculation was done every n samples. Every n th without stiffness means that only the entire P calculation was done every n samples. Page 50.

List of Figures

- Figure 1.1. A 10 DOF FEM of a five-story building. Page 1.
- Figure 1.2. Algorithmic flow for system identification. Page 2.
- Figure 1.3. Flow of the flaw detection and identification algorithm using the GN search Page 9.
- Figure 2.1. FEM lattice for the 10 DOF model. Page 13.
- Figure 2.2. NTS structure. Page 18.
- Figure 2.3. Floor plan of the NTS structure. Page 20.
- Figure 3.1. Mean error for various states of measurement. Measured nodes are denoted by cross-hatching. The mean identification error is represented by the size of the red bar. Page 33.
- Figure 3.2. Simplified 10 DOF stick model. Page 27.
- Figure 3.3. Calculation errors for the 50 DOF problem with 5 observations. The errors are shown in the bar graph with the yellow bars denoting elements that were perturbed to 90% of their nominal value and blue bars denoting unperturbed elements. The green lines indicate where the measurements took place. Page 34.
- Figure 3.4. Calculation errors for the 50 DOF problem with 5 observations. The errors are shown in the bar graph with the yellow bars denoting elements that were perturbed to 90% of their nominal value and blue bars denoting unperturbed elements. The green lines indicate where the measurements took place. Page 35.
- Figure 3.5. Calculation errors for the 50 DOF problem with 5 observations. The errors are shown in the bar graph with the yellow bars denoting elements that were perturbed to 90% of their nominal value and blue bars denoting unperturbed elements. The green lines indicate where the measurements took place. Page 36.
- Figure 3.6. Power spectral densities of sensors 3, 6, 9, 12, and 15. Note the lack of power above 35 Hz for sensor 12. Page 37.
- Figure 4.1. Mean identification error for 10 DOF using Gauss-Newton method (red bars on left) and the EKF with sine input, $R = 0.1$, no added noise (blue bars). Page 51.
- Figure 4.2. Average identification error for the EKF with a -20 dB noise level. This is the first repetition of the 10 experiments above. The left (red) bar represents the mean

identification error with no averaging, while the right (blue) bar is the error with the last 100 samples averaged. It is clear that the sine wave input results in a more accurate identification. Page 52.

Figure 4.3. Average identification error for the EKF with a -20 dB noise level. This is the second repetition of the 10 experiments above. The left (red) bar represents the mean identification error with no averaging, while the right (blue) bar is the error with the last 100 samples averaged. It is clear that the sine wave input results in a more consistent performance. Page 53.

Figure 4.4. The actual values for five of the 25 stiffness value to be identified (blue), and the estimated values (red). These five values are next to observed nodes and should be easily identified. There were 100 samples analyzed. Note that the y axis is not the same scale for every plot. Page 54.

Figure 4.5. The change in error covariance P in percent for 1000 samples at 100 samples/second. Note the scale here is multiplied by 106, the maximum value being $3.5 \times 106\%$. Page 55.

Chapter 1. Introduction to the problem

The design of engineering systems has traditionally resulted from a mix of theory, empiricism, and experience based on historical performance. With the advent of high-performance computers and massively parallel computations, it is hoped that simulations based on first principles will play an even more commanding role in the engineering process. However, for this to come about, it is essential that large-scale simulation models yield demonstrably accurate results for the particular problem type under study. The ability to validate computational models, based on a rigorous comparison between the actual measured system response and the simulated response, is a requisite to building confidence toward a simulation-dominated engineering process.

This project is concerned with the evaluation and validation of computational structural models known as finite element models (FEM). Finite element models represent a continuous structure with one that is comprised of a finite number of discrete elements. Mechanical engineers have become quite skilled in the derivation of relationships between these finite elements so that the finite model can be an effective approximation of the continuous structure. The improvement of the model, however, depends on the individual engineer and his or her experience and talent.

The methodology under development provides a more systematic method of model improvement and relies on state-space model-based signal processing to compare simulations with measured structural response. The signal processing algorithms are intended to determine the degree to which the simulation model represents the actual behavior of the structure, and to provide guidance on how the simulation model could be improved. In addition, the signal processing can sense changes in the structural system, which offers the potential for establishing an approach for damage detection in large structural systems.

A simple, 10 degree of freedom (DOF) FEM of a five-story building is shown in Figure 1.1. Each floor is represented as a node with mass M , and the nodes are connected by single column

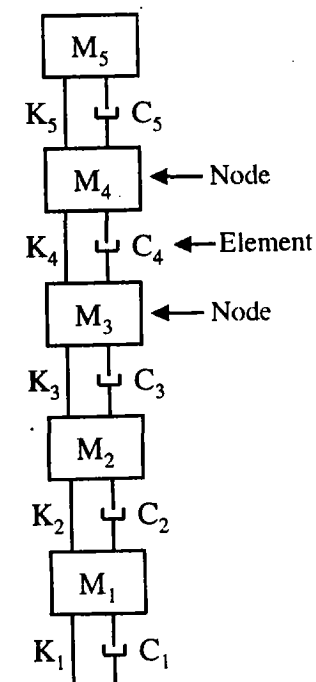


Figure 1.1. A 10 DOF FEM of a five-story building.

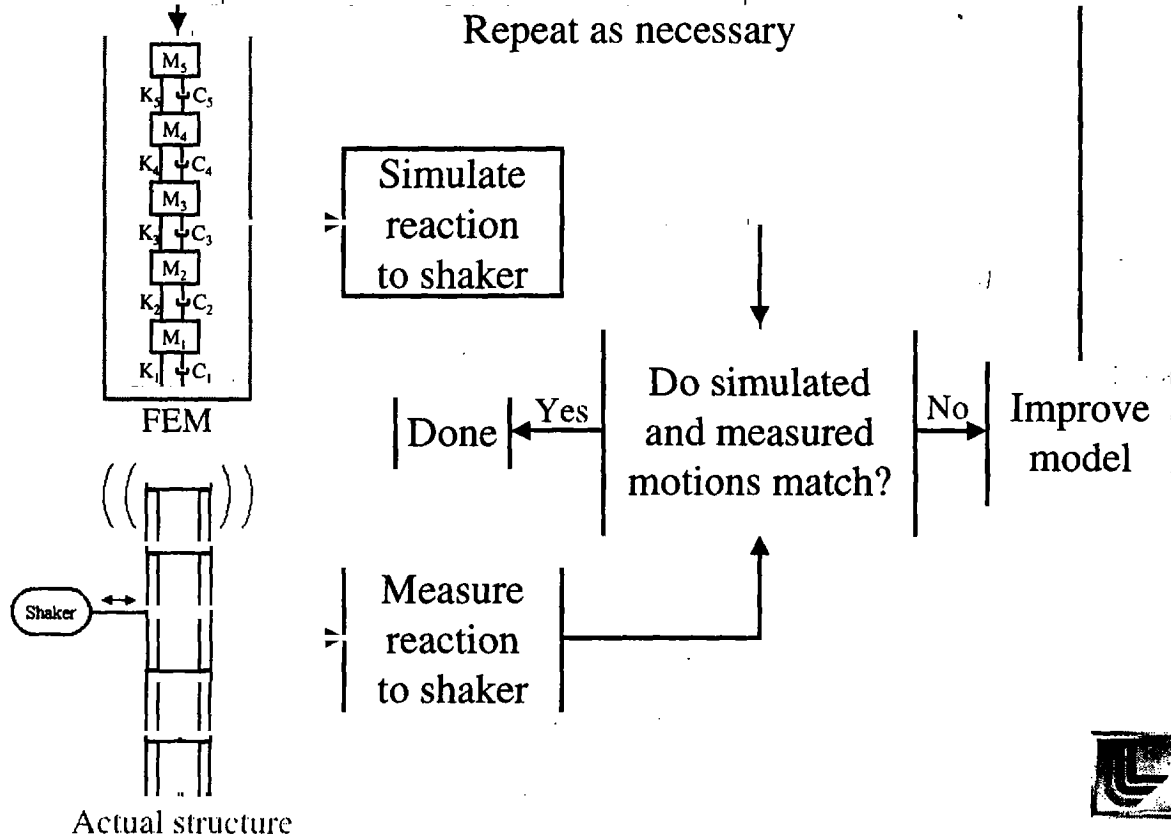


Figure 1.2. Algorithmic flow for system identification

elements of stiffness \mathbf{K} and damping \mathbf{C} . The stiffness elements \mathbf{K} are the ones to be identified, the mass and the damping values are considered constant. It is important to note that the nodes are where the measurements occur but the stiffness of the columns are the quantities to be identified.

The signal processing algorithm under development follows the procedure outlined in Figure 1.2. The essential inputs include the mass, damping, and stiffness matrices which are generated by the finite-element structural simulation model, and the measured response of the structure for a given loading. Ideally, the structure should be subjected to carefully controlled white noise excitations for the model identification process so that the input forcing function is precisely known and a wide range of vibrational modes are excited. In reality, most real structures will be too large to excite manually and we will have to rely on ambient vibrations as the excitation source. This situation will require a model of the ambient excitation source.

This model-based algorithm uses the finite-element-generated system matrices as a starting point, and constructs a linear state-space model of the system based on the second-order finite-element equations of motion. A Kalman filter-based residual whiteness test (Whiteness_Test.m) is used to detect any difference between the simulation model and the actual structure response. The whiteness test determines if the residuals (difference between the actual output and modeled output) contain any correlated information. If correlated information is present and all noise sources are assumed white, the model is not an adequate description of the system. This can be due either to model inadequacies or damage to an otherwise well-modeled structure. Another way to identify model mismatch is by comparing the recorded and simulated resonance locations, determined by calculating the eigenvalues of $(\mathbf{M}^{-1}\mathbf{K})$.

1.1 Gauss-Newton gradient search

If a model mismatch is detected through the whiteness test, several algorithms may be employed in an attempt to identify the discrepancies. We chose to focus on two that take quite different approaches to the problem. In the first, an iterative Gauss-Newton gradient search routine is used to minimize the residuals in the least squares sense. In one dimension, Newton's method can be visualized by plotting the function to be solved and then guessing one of the zero points. A more refined estimate is computed by drawing a line tangent to the function at our estimate x_c (c for current estimate) and determining where this line crosses the x axis. This distance is termed Δx , and the residual is $\Delta y = y_c = f(x_c)$. Thus

$$\hat{x} = x_c - \Delta x,$$

where \hat{x} is the new estimate and

$$f'(x_c) = \frac{\Delta y}{\Delta x} = \frac{f(x_c)}{\Delta x}$$

so that

$$\hat{x} = x_c - \frac{f(x_c)}{f'(x_c)}.$$

For a minimization problem, the function we want to solve is $f'(x)$, so the equation above becomes

$$\hat{x} = x_c - \frac{f'(x_c)}{f''(x_c)}$$

and for matrices this step becomes

$$\hat{x} = x_c - \frac{\mathbf{J}(x_c)}{\mathbf{H}(x_c)}$$

where \mathbf{J} is the Jacobian and \mathbf{H} is the Hessian of the matrix $\mathbf{F}(x_c)$ which we are trying to minimize.

In the present case, we use Newton's method to minimize a least-squares cost function that depends on θ , the vector of k values we are attempting to identify:

$$f(\theta) = \frac{1}{2} \mathbf{E}(\theta)^T \mathbf{E}(\theta) = \frac{1}{2} \sum_{i=1}^m \epsilon_i(\theta)^2$$

where \mathbf{E} is a function of the estimation error (residuals) and $\epsilon_i(\theta)$ is the i^{th} residual out of m outputs. The first derivative of $f(\theta)$ is defined by

$$\nabla f(\theta) = \frac{d}{d\theta} \frac{\mathbf{E}(\theta)^2}{2} = \mathbf{J}(\theta)^T \mathbf{E}(\theta)$$

Similarly, the second derivative of $f(\theta)$ is

$$\nabla^2 f(\theta) = \frac{d}{d\theta} \mathbf{J}(\theta)^T \mathbf{E}(\theta) = \mathbf{J}(\theta)^T \mathbf{J}(\theta) + \mathbf{H}(\theta)^T \mathbf{E}(\theta)$$

so that Newton's method appears as

$$\hat{x} = x_c - \frac{\mathbf{J}(\theta)^T \mathbf{E}(\theta)}{\mathbf{J}(\theta)^T \mathbf{J}(\theta) + \mathbf{H}(\theta)^T \mathbf{E}(\theta)}$$

This method converges quite quickly if the initial guess is not too far off and there are no local minima nearby. Its drawback is that the Hessian $\mathbf{H}(\theta)$ is quite expensive to obtain, and if the analytical form of $\mathbf{E}(\theta)$ is not available (as in this case) both \mathbf{J} and \mathbf{H} will have to be approximated using finite difference models or secant methods. This means on the order of n (where n is the model order) calculations for \mathbf{J} and $(3n^2 + n)/2$ calculations for \mathbf{H} .

In the Gaussian approximation to Newton's method, the Hessian is discarded and the iteration proceeds as

$$\hat{x} = x_c - \frac{\mathbf{E}(\theta)}{\mathbf{J}(\theta)}$$

The expense of the iteration is considerably reduced, but the performance of this method depends on the magnitude of \mathbf{HE} compared to \mathbf{JJ} . If \mathbf{HE} is much less than \mathbf{JJ} , this form closely approximates the pure Newton algorithm. This occurs when $\mathbf{E}(\theta)$ is linear in θ , or when the

innovation \mathbf{E} is small. If these conditions are not met, the Gauss-Newton method may converge slowly or not at all. More information about this algorithm may be found in Appendix 7.8, [1], and [2]. A flowchart of the damage detection and identification algorithm can be found in Figure 1.3 at the end of this chapter. For more on the damage detection algorithm, see Appendix 7.6, “Flaw detection and identification algorithm (5 DOF)”.

In summary, the Gauss-Newton method works well for systems that are reasonably well-modeled, with small residuals, that are not too large. For these systems it can converge quickly. However, for high DOF systems, there may be many local minima that the algorithm can get “stuck” in and the global minima may not be detected. Also, models with significant nonlinearities or models that are poor may result in non-convergence. Finally, the GN method is an offline or “batch” algorithm; it operates on all the recorded data at once. Thus the matrices involved in its calculations can get large very quickly, resulting in a high computational expense.

1.2 The augmented state vector continuous-discrete extended Kalman filter (EKF) system identification algorithm (whew!)

The second algorithm considered was the continuous-discrete Extended Kalman Filter, further details of which may be found in [3]. In this algorithm the underlying process is modeled as a continuous system, while the measurements are considered to occur at discrete times. The normal state vector of displacements and velocities is augmented (or extended) by the parameters to be identified. The results in a continuous state representation of the problem as

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), t) + \mathbf{w}(t) \quad (1)$$

$$\mathbf{y}[k] = \mathbf{h}(\mathbf{x}(k), k) + \mathbf{v}[k] \quad k = 0, 1, 2, \dots \quad (2)$$

where $\mathbf{x}(t)$ is the extended state vector, \mathbf{f} is the function describing the evolution of the state vector, \mathbf{h} is a matrix function describing the relation between the states and the measured output, and $\mathbf{w}(t)$ and $\mathbf{v}[k]$ are the state and measurement noise vectors with $\mathbf{Q}(t)$ and $\mathbf{R}[k]$ as the respective covariance matrices. $\mathbf{P}(t)$ is defined as the error covariance matrix, which we are attempting to minimize. In this sense the two algorithms are trying to minimize the same quantity, namely the covariance of the residuals. If this quantity can be minimized sufficiently,

the model may be assumed to be a good one. It is only the *method* of minimization that differs between the two.

For the continuous-discrete EKF, we use the following nomenclature:

$\hat{\mathbf{x}}$ = measurement vector

$\mathbf{x}[k_+]$ = state just after measurement

$\mathbf{x}[k_-]$ = state just before measurement

After each measurement, Equations 1 and 2 are linearized about the measurement and the states propagated to the next measurement time. A first order approximation is made for (1), and a second order approximation is made for the state covariance matrix $\mathbf{P}(t)$. This leads to

$$\dot{\mathbf{x}}(t) \approx \mathbf{f}(\hat{\mathbf{x}}, t) \quad (3)$$

$$\dot{\mathbf{P}}(t) \approx \mathbf{F}(\hat{\mathbf{x}}, t)\mathbf{P}(t) + \mathbf{P}(t)\mathbf{F}^T(\hat{\mathbf{x}}, t) + \mathbf{Q}(t) \quad t_{k-1} \leq t < t_k \quad (4)$$

where

$$\mathbf{F}_{ij} = \left. \frac{\partial f_i(\mathbf{x}, t)}{\partial x_j} \right|_{\mathbf{x}(t) = \hat{\mathbf{x}}(t)}$$

So before each measurement point, $\mathbf{P}[k_-]$ and $\mathbf{P}[k_-]$ are calculated according to 3 and 4 above.

To update the system following a measurement at time k , the following steps are taken:

- 1) Calculate the Kalman gain: $\mathbf{K}[k] = \mathbf{P}[k_-]\mathbf{H}_k^T(\hat{\mathbf{x}}[k_-]) \cdot [\mathbf{H}_k(\hat{\mathbf{x}}[k_-])\mathbf{P}[k_-]\mathbf{H}_k^T(\hat{\mathbf{x}}[k_-]) + \mathbf{R}[k]]^{-1}$
- 2) Update the state equation: $\hat{\mathbf{x}}[k_+] = \hat{\mathbf{x}}[k_-] + \mathbf{K}[k] \cdot [\mathbf{y}[k] - \mathbf{h}(\hat{\mathbf{x}}[k_-], k)]$
- 3) Update the state covariance: $\mathbf{P}[k_+] = [\mathbf{I} - \mathbf{K}[k]\mathbf{H}_k(\hat{\mathbf{x}}[k_-])]\mathbf{P}[k_-]$

where

$$\mathbf{H}_{ij} = \left. \frac{\partial h_i(\mathbf{x}, t)}{\partial x_j} \right|_{\mathbf{x}(t) = \hat{\mathbf{x}}(t)}$$

The states are now propagated according to Equations 3 and 4 to the next measurement point. It is clear that the measurement points are not required to be regular nor even to occur at all, but

given relatively noise-free measurements, the more that can be taken, the better the state estimation.

For a 1-D problem the states are defined as

$$\mathbf{x} = [x \quad \dot{x} \quad k]^T$$

with x representing the displacement of the SHO and k the stiffness. The equations describing their evolution are

$$\begin{aligned}\dot{x}_1 &= x_2 \\ \dot{x}_2 &= -\frac{x_1 x_3}{m} - \frac{b}{m} x_2 + \frac{u}{m} \\ \dot{x}_3 &= 0\end{aligned}\tag{5}$$

so that

$$\mathbf{F} = \begin{bmatrix} 0 & 1 & 1 \\ -x_3/m & -b/m & -x_1/m \\ 0 & 0 & 0 \end{bmatrix}$$

and

$$\mathbf{H} = [-x_3/m \quad -b/m \quad -x_1/m]$$

The only assumptions made in this algorithm are that the system and measurement noises are uncorrelated, the initial state is a Gaussian with the given initial state as the mean and P_0 as the variance. No assumption is made about the input u , although the more frequencies it contains (and thus the more modes it may excite) the better the identification.

In summary, the continuous-discrete EKF method is recursive, can operate on nonlinear problems, and can compensate for poorly-ordered models and measurement noise through the judicious use of $\mathbf{Q}(t)$ and $\mathbf{R}[k]$. If necessary, higher order nonlinear estimates can be employed to further refine the identification. However, solving continuous projections of the \mathbf{x} vector and \mathbf{P} matrix can be computationally intensive, model order mismatch can prevent convergence, and only white noise models accommodated easily. See Table 1.1 for a comparison between the two methods.

For information on the effects of the various noise parameters on the convergence time of the EKF in 1-D, see Appendix 7.15, for more general information see 7.14, 7.16, and 7.18. For more information on the Gauss-Newton algorithm, see Appendices 7.5, 7.6, 7.8, and 7.12.

Algo- rithm	Real-time possible?	Computational expense	Noise robustness	Accuracy with full measurements	Accuracy with few measurements	Handle non- lineaities?	High DOF?
GN	No	Heavy	Good	Excellent	Ok	Not well	No
EKF	Yes	Moderate	Good	Excellent	Good	Limited	Limited

Table 1.1. Relative qualities of the Gauss-Newton and the Extended Kalman Filter algorithms

References

1. Lennart Ljung (1987). "System Identification theory for the user". Prentice-Hall, Englewood Cliffs, NJ. ISBN 0-13-881640-9
2. J.E. Dennis and R.B. Schnabel (1983), chapter 10. "Numerical Methods for Unconstrained Optimization and Nonlinear Equations", Prentice-Hall, Englewood Cliffs, NJ. ISBN 0-13-627216-9
3. Gelb, A. (ed. 1999). "Applied Optimal Estimation", The M.I.T. Press, Cambridge, MA. ISBN 0-262-57048-3.

Flaw detection and identification algorithm

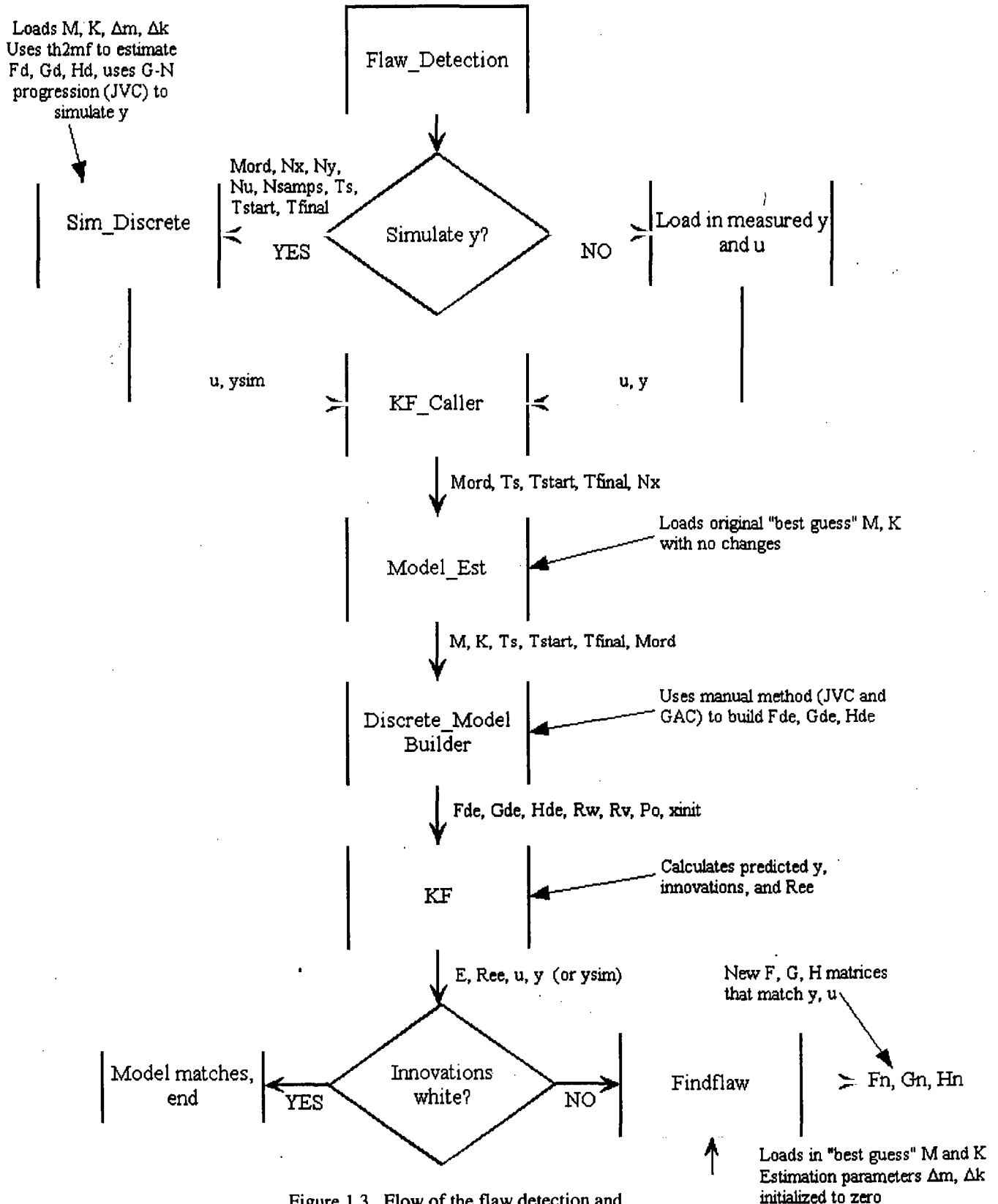


Figure 1.3. Flow of the flaw detection and identification algorithm using the GN search

Chapter 2. The Models

The models used in these experiments were finite element models supplied by the MEs in the form of a mass matrix \mathbf{M} and stiffness matrix \mathbf{K} . The damping matrix was constructed by using a linear combination of \mathbf{M} and \mathbf{K} , which will be discussed shortly.

The matrices \mathbf{M} and \mathbf{K} are stored by the ME programs as lists of ASCII numbers saved in some weird Unix form with names like KG.INFO and K.PTRS. There are 10 files in all, 5 that describe the \mathbf{K} matrix ($\mathbf{K}.*$) and 5 that describe the \mathbf{M} ($\mathbf{KG}.*$). So that we can use them, these are opened in Microsoft Word as "plain text" files (the default), and saved with the same name with ".txt" appended (again the default). This translates them to a readable .txt format for the PC.

A program with the structure `[M, C, K] = Files2MCK` is then called to translate the files. Before you use it, be sure to change the directory that the files are in on line 16 (this wasn't used enough to polish up very much). It in turn calls the real translation program, `Files2Matrices.m`. This program takes the gobbledygook that the MEs used and turns it into matrices that Matlab (and we) can understand. The same routine can be used to translate the ME acceleration data into vectors using `Files2accel.m` which calls `Files2Matsaccel.m`. The directory locations for both the input and output files have to be set within the calling program each time they are used. The translational programs do not need to be changed each time.

Using the above we can recover the matrices used for the different types of FEM we are interested in. They tend to have rounding errors present, but that isn't important as we have to build up the matrices on the fly in our identification programs as the number of independent variables is much smaller than the number of members in the matrices. We use the matrices given to us by the MEs only to check our model construction process.

2.1 Building \mathbf{K} given the identification parameters

Simply having the correct forms for \mathbf{M} and \mathbf{K} do us little good for identification purposes. The reason is that there is an underlying structure that dictates how the matrices are constructed according to how the FEM is calculated. The \mathbf{M} matrix is usually quite simple, but the \mathbf{K} matrix can be complex due to the cross-connection between members. For example, for a five DOF system the \mathbf{M} and \mathbf{K} matrices are as follows:

M =

5045	0	0	0	0
0	5045	0	0	0
0	0	5045	0	0
0	0	0	5045	0
0	0	0	0	3809.9

K =

3.2443e+006	-1.6222e+006	0	0	0
-1.6222e+006	3.2443e+006	-1.6222e+006	0	0
0	-1.6222e+006	3.2443e+006	-1.6222e+006	0
0	0	-1.6222e+006	3.2443e+006	-1.6222e+006
0	0	0	-1.6222e+006	1.6222e+006

However, there are actually only 5 independent variables in **K**. If we call them E_1, E_2, E_3, E_4 , and E_5 , **K** looks like this:

	$E_1 + E_2$	$-E_2$	0	0	0
	$-E_2$	$E_2 + E_3$	$-E_3$	0	0
K =	0	$-E_3$	$E_3 + E_4$	$-E_4$	0
	0	0	$-E_4$	$E_4 + E_5$	$-E_5$
	0	0	0	$-E_5$	E_5

It is clear that **K** is symmetric and that even though there are 13 members, symmetry and structure drops the number of independent variables to 5. This is also the correct number of unknowns for a 5 DOF system, so this is what we use as our identification parameters. If we use the symmetric members (of which there are 9), we are overspecifying the problem. This can prevent the algorithm from converging, but sometimes it will converge to an answer in which only the independent variables are changed and the dependent ones are unchanged. I will discuss this more in Chapter 4.

So now that we know there is internal structure to the **K** matrix, how do we construct it on the fly given the n independent variables ($n = \# \text{ DOF}$)? This is a more involved question that entails digging into the methods used by the MEs to construct the FEM. The files used to do this calculation are TH#DOF.M, where # is the DOF. These files in turn call more files (kcolumn.m,

kbeam.m, elem#.mat, nodes#.mat) which have to be prepared for each new model. I'll go over that process now.

The ME FEM representation

The way the MEs represent the FEMs that I have used in these experiments are like a lattice. Each connection point represents a node where different elements are joined. The same lattice can have many different DOF depending on the restrictions placed on the nodes. For example, the 10 DOF lattice is shown in Figure 2.1. There are 24 nodes and 35 elements, although there are only 10 DOF. The 10 DOF model is derived from the more complex lattice by specifying which nodes are fixed and which are free to move. In the 10 DOF model, all the nodes to the right of the first column are fixed to move with the nodes in the left-most column. For example, nodes 6, 7, and 8 are all "slaved" to node 5. Likewise nodes 14, 15, and 16 are all slaved to node 13. In essence, this reduces the 5 story, four column building to a 5 story, 1 column "stick figure" while keeping the structure physically realizable.

Decoding the ME files

In order to know how the models are constructed and keep track of which nodes are slaved to which other ones, the MEs use three files: `elem.txt` (describes the types of elements and how they are connected), `eqn.txt` (describes the nodes and their relationship to one another), and `readme.txt` (contains the specifications for the elements). These files are cryptic, and require decoding.

We'll start with `elem.txt`. It is contained in the icon on the right – just click to open (if you have a paper copy, it is appended to the end of this chapter). It contains a list with four columns. The first is the number of the element, the second is its type (in this case, type 1-5 are column elements, types > 5 are beam elements). Each type has its own specifications which are spelled out in the `readme.txt` file), and the third and fourth columns are the nodes to which the elements are attached. All of these elements are simple beams, they only attach at two nodes. For more complex elements more attachment points may be included. All of this information is used later in `th10dof.m` to construct the **M** and **K** matrices.



The information in `eqn.txt` is not so straightforward. It is also included on the right or at the end of this chapter. It has ten columns, the first of which is just the node number. The next three are the position of the nodes on the grid (usually in inches), and the next six tell which equations of motion are associated with what possible node motions. The first



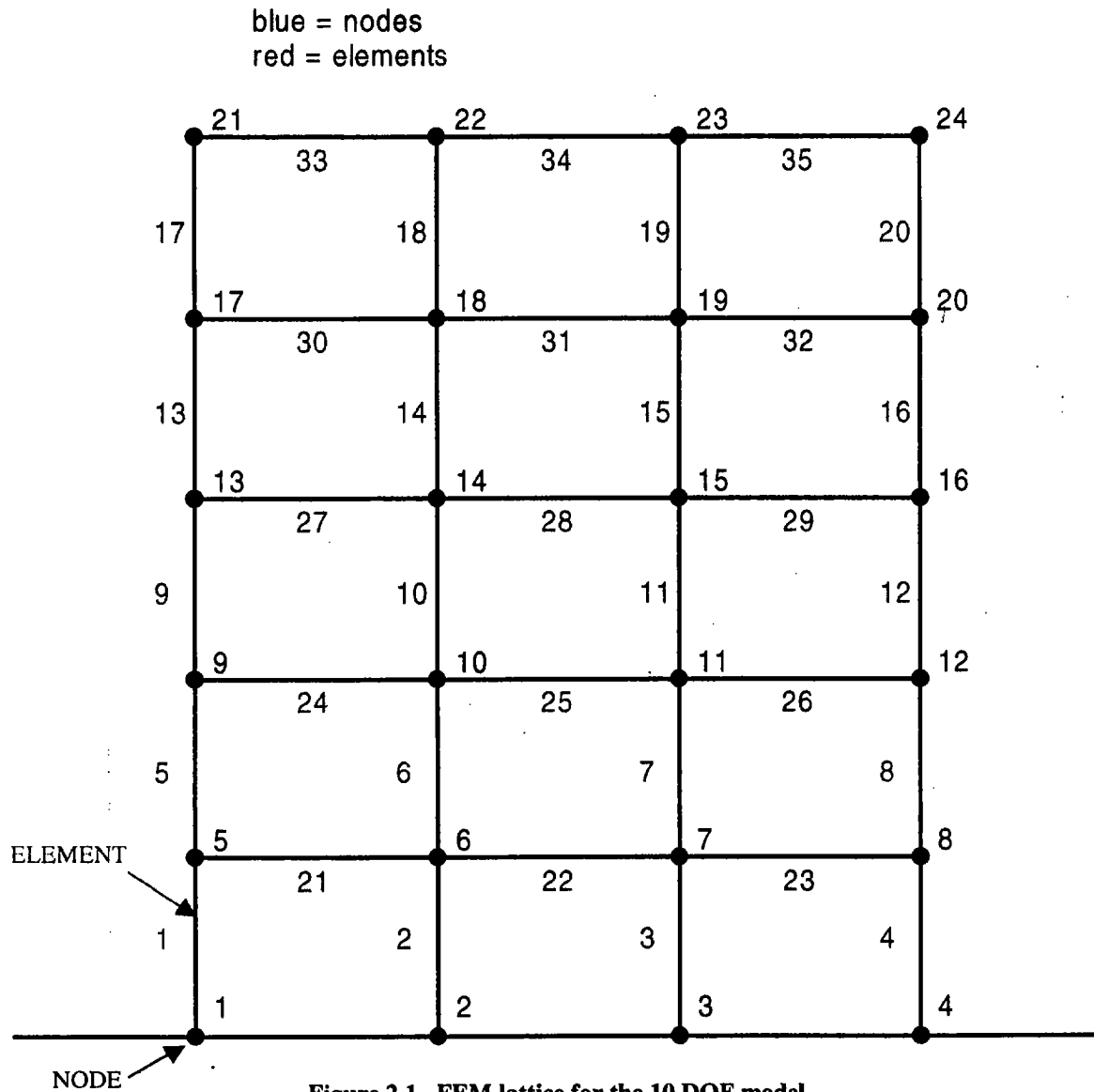


Figure 2.1. FEM lattice for the 10 DOF model.

three columns of the six are translational motion and the last three are rotational. For example, in the 10 DOF lattice above the first four nodes are set into the ground (fixed) and cannot move at all, so they have no equations of motion associated with them. Node 5, however, is free to move, but only in the linear x direction (eq- x) and the rotational z direction (eq- rz). These are assigned to equations 1 and 2, respectively. Nodes 6, 7, and 8 are assigned the same equations of motion and are therefore constrained to move exactly like node 5. This is how some nodes are restricted, lowering the total DOF of the model. Note that there are only 10 equations, corresponding to the 10 DOF.

Building the K matrix from the identification parameters

The **K** matrix we need to use in the identification process (**K** global) is composed of smaller **k** matrices (**k** local) added together. The 3D local **k** matrix is calculated by first constructing **k_local** in both `kbeam.m` and `kcolumn#.m`:

$$k_local = \begin{bmatrix} EA/L & 0 & 0 & -EA/L & 0 & 0 \\ 0 & 12EI_z/L^3 & 6EI_z/L^2 & 0 & -12EI_z/L^3 & 6EI_z/L^2 \\ 0 & 6EI_z/L^2 & 4EI_z/L & 0 & -6EI_z/L^2 & 2EI_z/L \\ -EA/L & 0 & 0 & EA/L & 0 & 0 \\ 0 & -12EI_z/L^3 & -6EI_z/L^2 & 0 & 12EI_z/L^3 & 6EI_z/L^2 \\ 0 & 6EI_z/L^2 & 2EI_z/L & 0 & -6EI_z/L^2 & 4EI_z/L \end{bmatrix}$$

where E = stiffness coefficient (parameter), A = cross-sectional area in in^2 = width*depth, L = length (inches), and I_z = rotational moment of inertia (all given, usually in a `readme.txt` file). For the 5, 10, and 50 DOF models the E that is multiplied by I is varied, while the E multiplied by the A is considered constant. Once **k_local** has been built given the latest estimate of the E values, it is transformed into the **k** matrix using a transformation matrix **T**:

$$k = T^T \cdot k_local \cdot T$$

where

$$T = \begin{bmatrix} c & s & 0 & 0 & 0 & 0 \\ -s & c & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & c & s & 0 \\ 0 & 0 & 0 & -s & c & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

where

$$c = \cos(\beta)$$

$$s = \sin(\beta)$$

and $\beta = \pi/2$ in this case. For each element type, the subroutines `kbeam#.m` and `kcolumn#.m`, where # is the DOF, must be constructed to accept the identification parameters and return the local **k** matrix. So far, though, `kbeam.m` is the same for all models.

Now that we have **k**, we need to know what members of it we should use to build **K** and how to add them together. This process is involved, but I'll do my best to explain it and then we'll look at an example. It might help to open up `th10DOF.m` and follow along at this point. I have imbedded it in the document for your convenience.



The information in `elem.txt` and `eqn.txt` are stored in `.mat` files called `elem10.mat` and `nodes10.mat`. In each `.mat` file is a matrix with the information contained in the text file.

For each element, the `elements` and `nodes` matrices are loaded in from the `elem10.mat` and the `nodes10.mat` files. The element type is determined using the second column of `elements`, and that is sent to `kbeam.m` or `kcolumn10.m` and the proper **k** matrix returned. The nodes that the element connects with are determined using the third and fourth columns of `elements`. The equations associated with the element can then be determined by examining columns 5-10 (I restricted it to only 5,6, and 10 in `th10DOF.m` to speed it up) of the nodes. These in turn give us the local and global indices through the `find` command. We then proceed to build **K** using **k** and the indices calculated using the `find` command. Let's take a look at one step of a calculation.

Example: Calculating one section of **K** due to one element

In this example we'll follow the calculation of the **K** contribution for element 5 in the 10 DOF model. We'll start at the element loop (`for m = 1:length(elements)`):

`eltype = 2`, so this is a column element, therefore we call `kcolumn10.m` and get

`k =`

```

4.0554e+005      0 -2.9199e+007 -4.0554e+005      0 -2.9199e+007
      0 1.3455e+007      0      0 -1.3455e+007      0
-2.9199e+007      0 2.8031e+009 2.9199e+007      0 1.4015e+009
-4.0554e+005      0 2.9199e+007 4.0554e+005      0 2.9199e+007
      0 -1.3455e+007      0      0 1.3455e+007      0
-2.9199e+007      0 1.4015e+009 2.9199e+007      0 2.8031e+009

```

and

```
nodes = [5 9]
```

```

nodeqn = [1  0    2    3    0    4]      (these are the equations
                                           associated with the nodes)
lind =   [1  3    4    6]  (local indices, these are the locations of nonzero equation #)
gind =   [1  2    3    4]  (global indices, these are the equation numbers)
li = length(lind);

```

Now, solve for all two-member permutations:

```
loc =
```

1	1
1	3
1	4
1	6
3	1
3	3
3	4
3	6
4	1
4	3
4	4
4	6
6	1
6	3
6	4
6	6

```
glo =
```

1	1
1	2
1	3
1	4
2	1
2	2
2	3
2	4
3	1
3	2
3	3
3	4
4	1
4	2
4	3
4	4

Finally, build **K**:

```

for i = 1:li^2
    K(glo(i,1),glo(i,2)) = K(glo(i,1),glo(i,2)) + k(loc(i,1),loc(i,2));
end

```

This builds up **K** from all its former versions. The final step is where roundoff errors are removed using the `flush.m` command. The roundoff errors are present because large ($\sim 10^8$) values are added and subtracted together and sometimes they don't quite equal zero when they should due to roundoff problems.

M is just considered constant and recalled from the original stored matrices. It is simple to make the members of **M** identification parameters (see `fiveDOF.m` for an example), but was not considered necessary as the mass of the different members can be described with pretty good accuracy and should not change much after a subtle damaging event.

Calculating C

Now that we have the **M** and **K** matrices, we need to calculate **C** to be able to parameterize the model completely. The way this is done is to form a linear combination of the **M** and **K** matrices according to the following method: First, the eigenvalues of the $\mathbf{M}^{-1}\mathbf{K}$ matrix are determined (the natural resonances of the system without damping). There will be one resonance per degree of freedom of the system. The first and fourth, ω_1 and ω_4 (in radians) are used with the estimate damping percentage to calculate the coefficients a_0 and a_1 :

$$a_0 = \frac{2d\omega_1\omega_4}{\omega_1 + \omega_4}$$

$$a_1 = \frac{2d}{\omega_1 + \omega_4}$$

so that finally

$$\mathbf{C} = a_0\mathbf{M} + a_1\mathbf{K}.$$

Summary of model construction

To summarize, for each model several files must be constructed. They are:

1. A `.mat` file that contains the **M** and **K** matrices as given by the MEs (i.e. `10MCK.mat`).
2. The `elements` and `nodem` matrices, from `elem.txt` and `eqn.txt`.
3. `kcolumn#.m` and `kbeam#.m`, where # is the DOF. Returns the local **k** matrix given the identification parameters.
4. `getpars#.m`, a function that returns the default initial estimate of the ID parameters.
5. `th#DOF.m`, builds the theta model for the system in Matlab. Just returns the **M**, **C**, and **K** matrices if used with the EKF (named `th#DOFMCK.m` in that case)

6. `obsmat#.mat`, a vector describing which nodes (really equations) are measured. A one is entered for observed equations, a zero for unobserved equations.

2.2 The simulated 5 story structure models (5, 10, 50, 400 DOF)

These models all describe a 5 story building with varying amounts of complexity. The 5 DOF model only allowed x-translational motion at the left five nodes, the 10 DOF allowed the same five translational motions as the 5 DOF but added z-rotational motions as well. It is considered a better model, so after its introduction the 5 DOF model was not used. The 50 DOF model was just the 10 DOF with each column subdivided into 5 sections, so there was a total of 25 nodes, each with a x-translational and z-rotational DOF and with different masses on the floor nodes and column nodes. The 400 DOF subdivided the columns into 10 pieces and also freed the nodes to the right to move on their own. This resulted in 200 nodes, each with 2 DOF.

2.3 The NTS 5 DOF structure

The NTS 5 DOF structure is a different animal entirely. In order to keep the numbers of DOF down (to help speed processing and debugging) this model is a shear model, which has different identification parameters from the simulated models above. Instead of E , we use the shear parameter α_y . This is because the model is completely insensitive to changes in E , but quite sensitive to changes in α_y (see chapter 4.2 for a discussion of this). Other than that, the way that the model is constructed from the α_y parameters is the same. The only difference is in the calculation of k in `kcolumnNTS5.m`.

2.3.1 The NTS data

A large physical model of a five-story building was available for our use at the Nevada Test site, so on February 24, 2000 we went there to instrument and record its response to both sinusoidal and white noise excitations. The people present were myself, Dave McCallen, Matt Hoehler, and Tom Woehrle. The model (seen in



Figure 2.2. NTS structure.

Figure 2.2) was built with steel pipes and shelves, was 14 feet high, with floors that were about 1 square foot in size weighted with lead bricks. It was excited on the third floor via a stinger from a freestanding shaker supported from a sling. The stinger was as on-

Channel #	Location	Direction	Accelerometer type	Sensitivity (mV/g)
1	CL 3 rd floor	+y	LC	103.5
2	1 st floor	+x	63A 172	449
3	1 st floor	+y	63A 172	431
4	1 st floor	+x	3134	500
5	2 nd floor	+x	63A 120	478.6
6	2 nd floor	+y	63A 120	479.5
7	2 nd floor	+x	3134	500
8	3 rd floor	+x	63A 184	456
9	3 rd floor	+y	63A 184	424
10	3 rd floor	+x	3134	500
11	4 th floor	+x	63A 160	425.7
12	4 th floor	+y	63A 160	424.1
13	4 th floor	+x	3134	500
14	5 th floor	+x	63A 113	431.4
15	5 th floor	+y	63A 113	419
16	5 th floor	+x	3134	500

Table 2.1. Data for the accelerometers from the NTS experiments.

center as possible in order to reduce torsional modes and emphasize translational ones. A drawing of one of the floors appears in Figure 2.3. The points u and t are associated with the accelerometer mounting, for more information ask Dave McCallen.

The accelerometers used and their characteristics are shown in Table 2.1. I don't remember why there were two +x accelerometers per floor, as I have only ever worked with the signals in the +y direction, in line with the excitation. The sensitivity was used to convert between volts and g's. However, the data were recorded in 12-bit bins, from -2048 to +2048. The maximum voltage allowed (see Table 2.2) was varied in each experiment in order to get the best resolution. The samples to g conversion sequence was as follows (found in `NTS2ascii.m`):

$$\text{Voltage correction factor (VCF)} = \text{max voltage} / 2048 \quad (\text{as per channel and experiment, see Table 2.2})$$

Sensor calibration (SC) = V/g (in column 5 of Table 2.1)

$g \text{ data} = \text{sample data} * \text{VCF} * \text{SC}$

The data stored in the .mat files is in g's as calculated above. However, unbeknownst to me at the time, this was in English unit g's, to translate to inches per second squared we still have to multiply by 386.4. This is done after the data are read from the .mat file in files like EKF_NTS.m.

There were two types of inputs used: a swept sinusoid and white noise. The swept sine was started at 3 Hz and went up to 50 Hz in $\frac{1}{4}$ Hz steps, staying on each frequency for 50 cycles. Therefore the lower frequency excitations took the longer time. Each swept sine experiment took several minutes to record. The white noise was supposed to contain frequencies between 0.25 Hz and 51 Hz, but in actuality went from about 5 to about 55 Hz (see Figure 9 of Appendix 7.20).

Each white noise input file was recorded midstream (after any transient vibrations associated with startup had dissipated) for 60 seconds. The sampling frequency was 400 Hz. The filenames were done in the following manner:

$$\text{struc} \begin{pmatrix} o \\ d \\ b \end{pmatrix} \begin{pmatrix} w \\ s \end{pmatrix} \begin{pmatrix} \text{channel} \\ \# \end{pmatrix} \begin{pmatrix} \text{repetition} \\ \# \end{pmatrix} \text{bin}$$

where o/d/b denotes either original, damaged (the two inner pipes were removed to simulate damage), or background noise and w/s either the white or swept sine input. Thus strucow3x5.bin is the original structure, with white noise, the third channel, and the fifth repetition. The data is located on the P3-450 machine under H:\Data\Structures.

The .bin files were read into Matlab and saved as .mat files, 16 files per .mat file. They were saved using the same convention as above but omitting the channel # and the spacer x.

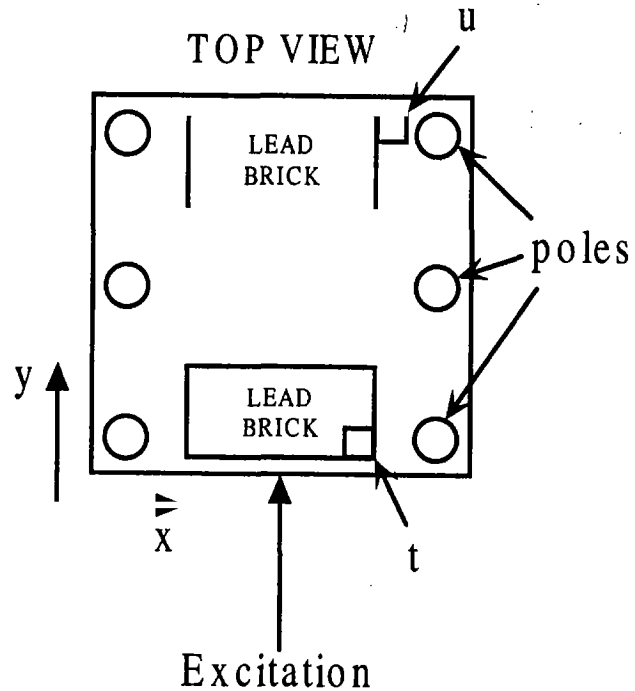


Figure 2.3. Floor plan of the NTS structure.

Thus `strucds1.mat` is the data from the damaged structure, swept sine input, first repetition. They contain `u1` (the input, channel 1), and `y1` through `y15` (the outputs, channels 2-16). The .mat files are located in `H:\Data\Structures\Mats`.

Experiments

The experiments went as follows, # represents the channel number:

Exp #	Filenames	Reps	V_{\max} ch. 1	V_{\max} ch. 2-16
1	<code>strucos#x1.bin</code>	1	+– 1.0 Volts	+– 1.0 Volts
2	<code>strucow#x1:2.bin</code>	2	+– 1.0 Volts	+– 1.0 Volts
3	<code>strucow#x3:5.bin</code>	3	+– 1.0 Volts	+– 0.25 Volts
4	<code>strucos#x2.bin</code>	1	+– 2.5 Volts	+– 2.5 Volts
5	<code>strucds#x1.bin</code>	1	+– 2.5 Volts	+– 2.5 Volts
6	<code>strucdw#x1:5.bin</code>	5	+– 1.0 Volts	+– 0.25 Volts
7	<code>strucdb#x1:5.bin</code>	5	+– 0.1 Volts	+– 0.1 Volts

Table 2.2. Filenames and maximum voltage limits for the NTS experiments.

Comments:

1. First attempt, all channels +–1.0 V, some clipping
2. 2 recordings of white noise input, +– 1.0 V.
3. 3 more recordings, decreased V_{\max} to +– 0.25 V for channels 2:16 to increase resolution
4. Redid of # 1 because of the clipping
5. Damaged swept sine input
6. Damaged white input
7. Background noise measurement, all channels +– 0.10 V

Elem.txt for 10 DOF

BEAM ELEMENT data

=====

m,	mtyp	node1	node2
1	1	1	5
2	1	2	6
3	1	3	7
4	1	4	8
5	2	5	9
6	2	6	10
7	2	7	11
8	2	8	12
9	3	9	13
10	3	10	14
11	3	11	15
12	3	12	16
13	4	13	17
14	4	14	18
15	4	15	19
16	4	16	20
17	5	17	21
18	5	18	22
19	5	19	23
20	5	20	24
21	6	5	6
22	6	6	7
23	6	7	8
24	7	9	10
25	7	10	11
26	7	11	12
27	8	13	14
28	8	14	15
29	8	15	16
30	9	17	18
31	9	18	19
32	9	19	20
33	10	21	22
34	10	22	23
35	10	23	24

Eqn.txt for 10 DOF

node	x	y	z	eq-x	eq-y	eq-z	eq-rx	eq-ry	eq-rz
1	0.00E+00	0.00E+00	0.00E+00	0	0	0	0	0	0
2	2.40E+02	0.00E+00	0.00E+00	0	0	0	0	0	0
3	4.80E+02	0.00E+00	0.00E+00	0	0	0	0	0	0
4	7.20E+02	0.00E+00	0.00E+00	0	0	0	0	0	0
5	0.00E+00	1.44E+02	0.00E+00	1	0	0	0	0	2
6	2.40E+02	1.44E+02	0.00E+00	1	0	0	0	0	2
7	4.80E+02	1.44E+02	0.00E+00	1	0	0	0	0	2
8	7.20E+02	1.44E+02	0.00E+00	1	0	0	0	0	2
9	0.00E+00	2.88E+02	0.00E+00	3	0	0	0	0	4
10	2.40E+02	2.88E+02	0.00E+00	3	0	0	0	0	4
11	4.80E+02	2.88E+02	0.00E+00	3	0	0	0	0	4
12	7.20E+02	2.88E+02	0.00E+00	3	0	0	0	0	4
13	0.00E+00	4.32E+02	0.00E+00	5	0	0	0	0	6
14	2.40E+02	4.32E+02	0.00E+00	5	0	0	0	0	6
15	4.80E+02	4.32E+02	0.00E+00	5	0	0	0	0	6
16	7.20E+02	4.32E+02	0.00E+00	5	0	0	0	0	6
17	0.00E+00	5.76E+02	0.00E+00	7	0	0	0	0	8
18	2.40E+02	5.76E+02	0.00E+00	7	0	0	0	0	8
19	4.80E+02	5.76E+02	0.00E+00	7	0	0	0	0	8
20	7.20E+02	5.76E+02	0.00E+00	7	0	0	0	0	8
21	0.00E+00	7.20E+02	0.00E+00	9	0	0	0	0	10
22	2.40E+02	7.20E+02	0.00E+00	9	0	0	0	0	10
23	4.80E+02	7.20E+02	0.00E+00	9	0	0	0	0	10
24	7.20E+02	7.20E+02	0.00E+00	9	0	0	0	0	10
25	-1.00E+01	0.00E+00	0.00E+00	0	0	0	0	0	0

Chapter 3. The Gauss-Newton algorithm

The Gauss-Newton algorithm, discussed in Chapter 1.1, is an iterative gradient search method of finding the minimum value of a function. Here we will discuss the results of our attempts to use the GN algorithm to identify the different models discussed in Chapter 2.

3.1 Simulated models

5 and 10 DOF, fully measured, no noise

For the 5 and 10 DOF simulated models with all translational nodes measured (there is no way to measure the rotational DOF), and with no noise added to the simulated measurements, the GN algorithm worked quite well. It would typically converge in only a few iterations, the number usually corresponding with the size of the model mismatch. That is, if the estimates of the ID parameters was quite poor, it would require more iterations to converge, as expected. The convergence time and accuracy of the final result would also depend highly upon the tolerance chosen, which controls how small the error can be before the algorithm terminates. For a large perturbation (50%, so the estimate of the stiffness was twice as large as the actual value) on one floor, the algorithm would converge in a single iteration with 3-8% accuracy with a tolerance of 0.1, and would take 6-13 iterations with 0.4-0.9% accuracy for a tolerance of $1e^{12}$. For changes on more than one floor of varying amplitudes the result was the same. In this region, the GN algorithm performs quite well.

10 DOF, not fully measured, no noise

The GN algorithm did not perform as well here as it was hoped. One reason is that as it uses a constant Kalman gain, it is essentially a Wiener filter that requires measurement of every node. As a result, I had to modify it so that it thought it was getting measurements at every node. There are two ways to do this: One, use the modeled outputs as the measured ones, thus forcing the residuals to zero for those particular nodes. Since they are zero, there will be no change calculated for the nodes that aren't measured. This seems to work well, but its effect on the accuracy of measured nodes is not well understood. The second is to "collapse" the matrices used in calculating the gradient so that only rows that correspond to measured nodes are included in the gradient calculation. Then the updates that are calculated are only applied to the measured nodes. This is more computationally efficient, as a 50 DOF system with only 5 measurements takes about the same amount of time to compute as a 5 DOF system. It does require slightly

more overhead, but speeds up the computation of large systems substantially. At this point it is important to keep in mind that we are still assuming that we can somehow derive the position of the nodes, either from the acceleration or from direct measurement. This was because in order to update the state equations we had to have information on the states, which are displacement and velocity. This was to cause problems, as we shall see.

The results for what I call my “standard perturbation” (the actual parameters are 7/8, 3/4, 9/10, 1/3, and 2/3 of the estimated parameters) are shown in Figure 3.1 at the end of this chapter. A summary of the parameters used were as follows:

$k = [7/8, 3/4, 9/10, 1/3, 2/3] * k_0;$	% all members of parameters varied, some significantly
No g restraints	% no restrictions on search vector magnitude
$\text{Pars}_0/100 < \text{pars} < 2 * \text{pars}_0$	% parameters (theta vector) restrained
Kalman gain = eye(ra,r)	% Kalman gain is the identity matrix (default, not optimal)
No noise, determinant criterion	% original algo used trace criterion

Measured nodes are denoted by cross-hatching, and the mean identification error is represented by the size of the bar on top of each model. The results are not too surprising – since the most poorly modeled states are 4 and 5, if they are both not measured the errors are usually high: 32, 47, 43, 179% errors. However, there was one experiment where states 3, 4, and 5 were not measured and the mean error was only 4.7%. This is better than when only states 4 and 5 were not measured – 32%! This shows that for the G-N algorithm, sometimes measurements can actually hurt accuracy, depending on the model and the size of the mismatches.

Another large error was when states 1,2, and 4 were not measured (89%). When the algorithm was run again with state 3 not measured, the mean error improved to 11%. Again, measurement of state 3 can be detrimental to performance. It could be because of the large change in k between the third and fourth floors, or just due to the inconsistencies of the G-N algorithm.

Overall, the results show that good to excellent (error rates from 0.02% to 0.67%) results for three measurements with the exception of states 4 and 5 unmeasured (32%, expected due to the large changes in k for elements 4 and 5). For two measurements there was much poorer performance (0.28, 4.69, 18.3, 26.8, 80.5, and 88.8% error) and for a single measurement the

best we could do was 6.6% (state 4 measured) and the worst was 179% (state 3 measured). This seems to indicate that it is possible to model the structure relatively well with only one measurement if *it is taken in the right place*. In this case, state 4 was very poorly modeled (actual value only 1/3 of modeled value) and so a measurement there helped to converge the solution. On the other hand, state 3 was modeled the best (actual value 90% of modeled value) and so a measurement there was not always helpful – in some cases, it was actually detrimental to performance.

50 DOF, full and partial measurement, no noise

In this experiment, the columns of the previous five-story building were discretized into five sections apiece, each with a translational and rotational DOF. The five floor nodes were always measured, and the location and size of the model mismatch was varied. Unlike the 10 DOF, for 50 DOF the GN algorithm does not converge perfectly to the actual structure in the absence of noise even with all 25 (not just the 5 floors) translational nodes measured. The best we can usually do (for small changes in k such as 5-10%) is converge to a mean error on the 25 translational degrees of freedom of about 5%, far worse than the 10 DOF case. We will discuss possible causes of this inaccuracy later.

In the following simulations all perturbations to the stiffness elements were to reduce the stiffness (EI) constant by only 10% to 90% of its nominal value, a small perturbation that the GN algorithm should converge to easily as the residuals should be small. In the first set of simulations all nodes are assumed measured, and in the second simulation only the floors (nodes 5:5:25) are measured. Keep in mind that the things that are measured are nodes, while the algorithmic parameters are the stiffness values for the elements, which are attached to other elements at the nodes. The structure is arranged so that element 1 is between nodes 1 and 2. Thus to observe node 5 is to be able to affect elements 1 and 5 (see Figure 2.1). However, the nodes as given in Figure 2.1 do not match up with the equation number, and can be confusing as there are many more nodes than equations. For simplicity I sometimes assume there are only 5 nodes, each with two DOF. This is a stick model with the lowest (fixed) node not counted. Thus node 1 is between elements 1 and 2, and node 5 is at the top by itself. See Figure 3.2 for a representation of the stick model. The context should make it clear if I am referring to the stick or the ME model.

The algorithmic details are:

No g restraints, Kalman gain = default

$\text{Pars}_0/100 < \text{pars} < 2 * \text{pars}_0$ % parameters (theta vector) restrained

No noise, trace criterion

Results (all translational nodes measured):

Ideally, the algorithm should converge to the answer with a negligible error since we are not including any noise in the process. However, the identification error varied from 0.3% to more than 30%, without any obvious trend to the accuracy. Clearly more work is needed to improve the behavior of the GN algorithm when faced with large DOF systems.

Results (only 5 measurements):

The results were not too surprising, in that damage to elements connecting unmeasured nodes was not readily calculable, with nodes that are farthest from the measured nodes suffering the largest errors. The unmeasured node damage was reflected in the measured node calculated damage. The results are shown in Figures 3.3, 3.4, and 3.5, at the end of this chapter. I have observed the following:

1. The problem of convergence is not simply an observational problem. This is illustrated by the first and final plots of Figure 3.5, where we see that even when observed modes are perturbed, the results are only accurate to within about 7%. I believe this is a function of the search algorithm, as right now the update is calculated as

$$\theta = \theta_0 + \lambda g$$

where θ is the parameter vector, g is the search direction, and λ is a constant. If all but one or two of the parameters are near their optimal value, the value of λ becomes very small, making convergence to the correct θ unlikely. I tried making λ a vector as well, which gives the algorithm the ability to tailor the update vector so that systems with many

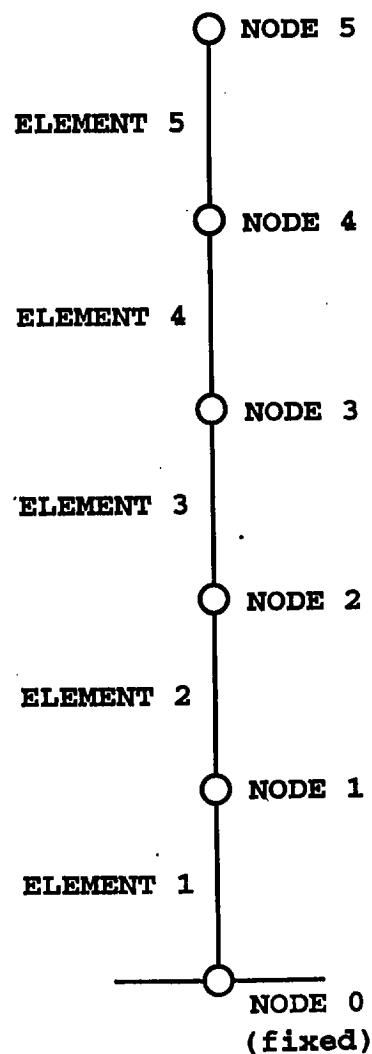


Figure 3.2. Simplified 10 DOF stick model

degrees of freedom can be minimized (`pem_sparse_vector.m`). This was unsuccessful as well, indicating that perhaps for 50 DOF the gradient search may be ineffective – the function that we are attempting to minimize may have too many local minima.

2. The farther away from the observed nodes, the poorer the convergence. This is expected, as the farther we move away from the measured nodes the less information we have about the actual state of the system. This cannot be avoided in the present incarnation, but the effects of the perturbations on the unmeasured nodes can be noticeable on the measured nodes. For example, in the fourth plot from the top of Figure 3.5, element 8 was reduced to 0.90 of its normal value. The system was unable to reduce element 8 to its correct value as it was not measured. However, to compensate, the values for elements 5-6 and 10-11 (the four elements closest to measured nodes, which the algorithm can change easily) were increased by 4.0, 3.7, 3.7, and 2.7% respectively. These are the largest errors of elements text to measured nodes in the system, and reflect the perturbation of the element located between them. This phenomenon did not always occur each time significant perturbations were made to the structure between measured nodes, but it could be quite useful as a diagnostic tool and for generally locating model mismatches in large models.
3. The convergence is still not good for many perturbations. In Figure 3.5, the first and second plot, it is clear that the convergence is only good near nodes 10 and 15. Near the others the error is still large. This would seem to indicate that the best convergence occurs in the middle of the structure. It also demonstrates that if many elements are perturbed or poorly modeled, the entire structure will not converge readily.

Conclusions for the simulated systems

For the simulated systems, the GN algorithm worked quite well if there was no measurement noise, the DOF was 10 or below, and all of the translational nodes were measured. If full translational measurement did not occur, the identification accuracy can vary widely, depending on the strengths and weaknesses of the model.

3.2 NTS 5 DOF model

Processing the NTS data introduces a host of difficulties not encountered when operating in the comfortable simulated world. The measurements are accelerations, not displacement, and they are noisy, with both low frequency drift and random additive noise. Also, the real NTS

structure, a 14 foot high steel-and-lead model, has more than a few DOF and oscillates in 3-D. The challenges this presented were many and varied.

Working with the data

In the algorithm above, the states are assumed to be the position and velocity of each node. This allows a convenient description of the problem in state-space. Our data, however, is measured in acceleration. Originally (and somewhat naively) it was believed that we could simply filter the data with a 1 Hz highpass filter and then use a perfect integrator twice to convert to displacement, as displacement is used as the state of choice during our 5 and 50 DOF simulation trials. However, there is some noise in the signals, which for the moment we will consider white. Larry pointed out that if you integrate a noise signal with a unity frequency spectrum, you get a noise spectrum of $1/s$. Do it again and now the noise spectrum is $1/s^2$, indicating that the noise has been significantly “reddened”, the noise at low frequencies has been increased. Depending on the noise level, it is possible to significantly distort the calculated displacement signal. Also, our modeling assumes white noise and will not operate as efficiently for reddened noise. Thus, we must either find another way to convert the acceleration data to position or change the G-N algorithm to work with acceleration, not displacement. I chose the latter.

Using acceleration instead of displacement

The first step in changing the algorithm to work with acceleration was to change the way the innovations are calculated. For this problem, modeled as a series of simple harmonic oscillators, the equations of motion are

$$\mathbf{M}\ddot{\mathbf{y}} + \mathbf{C}\dot{\mathbf{y}} + \mathbf{K}\mathbf{y} = \mathbf{u}$$

so that acceleration is related to velocity and displacement by

$$\ddot{\mathbf{y}} = \frac{\mathbf{u}}{\mathbf{M}} - \frac{\mathbf{C}_s}{\mathbf{M}} \dot{\mathbf{y}} - \frac{\mathbf{K}_s}{\mathbf{M}} \mathbf{y}$$

where \mathbf{M} , \mathbf{C}_s , and \mathbf{K}_s are the system mass, damping and stiffness matrices respectively. This means that the innovation is now represented by

$$\epsilon[n] = y[n] - \left[\frac{u[n]}{\mathbf{M}} - \frac{\mathbf{C}_s}{\mathbf{M}} \dot{y}[n] - \frac{\mathbf{K}_s}{\mathbf{M}} y[n] \right]$$

since the measurement $y[n]$ is now acceleration. The criterion is unchanged.

However, this is not the only change. Indeed, the calculation of \hat{x} has changed significantly due to the redefinition of the innovation. For displacement, from above:

$$\begin{aligned}\hat{x}[n+1] &= \mathbf{A} \cdot \hat{x}[n] + \mathbf{B} \cdot u[n] + \mathbf{K} \cdot \epsilon[n] \\ \hat{y}[n] &= \mathbf{C} \cdot \hat{x}[n]\end{aligned}$$

and for acceleration:

$$\begin{aligned}\hat{x}[n+1] &= \mathbf{A} \cdot \hat{x}[n] + \mathbf{B} \cdot u[n] + \mathbf{K} \cdot \left(y[n] - \left[\frac{u[n]}{\mathbf{M}} - \frac{\mathbf{C}_s}{\mathbf{M}} x_v[n] - \frac{\mathbf{K}_s}{\mathbf{M}} x_d[n] \right] \right) \\ \hat{y}[n] &= \frac{u}{\mathbf{M}} - \frac{\mathbf{C}_s}{\mathbf{M}} \dot{y} - \frac{\mathbf{K}_s}{\mathbf{M}} y = -[\mathbf{M}^{-1}\mathbf{K}_s \quad \mathbf{M}^{-1}\mathbf{C}_s] \cdot \hat{x}[n] + \mathbf{M}^{-1} \cdot u[n]\end{aligned}$$

where $y[n]$ is now an acceleration measurement and $x_d[n]$ are the displacement states and $x_v[n]$ are now the velocity states. Defining

$$\begin{aligned}x_d &= \mathbf{C}_d x \\ x_v &= \mathbf{C}_v x\end{aligned}$$

where \mathbf{C}_d is the same as our old displacement \mathbf{C} , we can write the above as

$$\begin{aligned}\hat{x}[n+1] &= \left[\mathbf{A} + \frac{\mathbf{K}\mathbf{C}_s\mathbf{C}_v}{\mathbf{M}} + \frac{\mathbf{K}\mathbf{K}_s\mathbf{C}_d}{\mathbf{M}} \right] \cdot \hat{x}[n] + \left[\mathbf{B} - \frac{\mathbf{K}}{\mathbf{M}} \right] \cdot u[n] + \mathbf{K} \cdot y[n] \\ \hat{y}[n] &= -[\mathbf{M}^{-1}\mathbf{K}_s \quad \mathbf{M}^{-1}\mathbf{C}_s] \cdot \hat{x}[n] + \mathbf{M}^{-1} \cdot u[n]\end{aligned}$$

or

$$\begin{aligned}\hat{x}[n+1] &= [\mathbf{A} + \mathbf{K}\mathbf{D}\mathbf{C}_s\mathbf{C}_v + \mathbf{K}\mathbf{D}\mathbf{K}_s\mathbf{C}_d] \cdot \hat{x}[n] + [\mathbf{B} - \mathbf{K}\mathbf{D}] \cdot u[n] + \mathbf{K} \cdot y[n] \\ \hat{y}[n] &= -[\mathbf{D}\mathbf{K}_s \quad \mathbf{D}\mathbf{C}_s] \cdot \hat{x}[n] + \mathbf{D} \cdot u[n]\end{aligned}$$

so that we may now write

$$\hat{x} = \text{litr} \left(\mathbf{A} + \mathbf{K}\mathbf{D}\mathbf{C}_s\mathbf{C}_v + \mathbf{K}\mathbf{D}\mathbf{K}_s\mathbf{C}_d, [\mathbf{K} \quad \mathbf{B} - \mathbf{K}\mathbf{D}] \begin{bmatrix} y \\ u \end{bmatrix}, \mathbf{x}_0 \right)$$

with $\mathbf{D} = \mathbf{M}^{-1}$.

Now that we have \hat{x} , we can proceed to the calculation of the G-N gradient, which is

$$\begin{aligned}\psi(n|\theta) = \frac{d(\hat{y}(n|\theta))}{d\theta} &= (-[\mathbf{D}\mathbf{K}_s \quad \mathbf{D}\mathbf{C}_s] \psi_x(n|\theta) + \mathbf{D} \psi_u) + \\ &\quad (-[\mathbf{D}\mathbf{K}'_s + \mathbf{D}'\mathbf{K}_s \quad \mathbf{D}\mathbf{C}'_s + \mathbf{D}'\mathbf{C}_s] \cdot \hat{x}[n] + \mathbf{D}' \cdot u[n])\end{aligned}$$

where ψ_x and ψ_u are the derivatives of \hat{x} and u with respect to θ and \mathbf{X}' denotes the derivative of the matrix \mathbf{X} with respect to θ . Since the input u is obviously not dependent on θ that term can safely be ignored. What is left now is to calculate ψ_x , the derivative of \hat{x} (Eq. 3).

$$\begin{aligned}\psi_x[n+1] = & [A + KDC_s C_v + KDK_s C_d] \cdot \psi_x[n] \dots \\ & + [A' + K'DC_s C_v + KD'C_s C_v + KDC'_s C_v + K'DK_s C_d + KD'K_s C_d + KDK'_s C_d] \cdot \hat{x}[n] \dots \\ & + [B' - K'D - KD'] \cdot u[n] + K' \cdot y[n]\end{aligned}$$

This can be calculated using LTITR:

$$\psi_x = \text{ltitr} \left[\begin{array}{c} A + KDC_s C_v + KDK_s C_d, \\ A' + K'DC_s C_v + KD'C_s C_v + KDC'_s C_v + K'DK_s C_d + KD'K_s C_d + KDK'_s C_d \dots \\ \dots K' \quad B' - K'D - KD', \end{array} \right] \begin{bmatrix} x \\ y \\ u \end{bmatrix}, dx_0$$

After all these additions are made, the algorithm is almost ready to run.

The search for K

As mentioned in Chapter 1, in the GN algorithm the Kalman gain K is calculated by

$$K = PC^T R_v^{-1}$$

where P is the state covariance matrix, C is defined above, and R_v is the measurement noise covariance matrix. However, this assumes P is constant, not always a good approximation. The Kalman filter algorithm normally does not assume P is constant, and calculates a new, corrected P at each time step:

$$P = (I - KC)P_p,$$

where I is the identity matrix and P_p is the predicted measurement for the time step. Thus K can be expressed as:

$$K = P_p C^T (C P_p C^T + R_v)^{-1}$$

However, we are still faced with the problem of what to use for P_p and R_v . The equation above is used at each time step in a Kalman filter algorithm, and LTITR.M does not do that. It is only an LTI kernel propagator, and assumes the K it is given is constant.

So really the only way to effectively implement the Kalman filter is to overhaul PEM_SPARSE.M so that it uses a full-fledged Kalman filter algorithm. This would emulate an optimal filter approach and almost certainly cause the accuracy to improve. This is the path chosen for the EKF algorithm.

Bad data on sensor 12?

I have noticed that the data from sensor 12 (fourth floor, +x direction), while of the correct magnitude, seems to be lacking in high frequency components. In Figure 3.6 the PSD of sensor 12 is compared with sensors 3, 6, 9 and 15, which were oriented the same way as sensor 12. It is clear that there are not as many high frequencies in sensor 12's spectrum.

Conclusions for NTS 5 DOF data

The simple algorithm PEM from Matlab has been substantially changed in order to operate on the NTS structure. The acceleration is used as the input to the system so that no transformation to displacement is required. However, the algorithm can be improved by implementing the full Kalman filter algorithm for state estimation, which we will explore in the next Chapter.

In addition, we may have to limit ourselves to analyzing data below about 35 Hz in order to compensate for sensor 12's inadequacies. It is not clear at this time if any of sensor 12's data is useful, we may have to try and update the model without it. As it is only one sensor, the loss should not affect the accuracy more than a few percent. I have tried both lowpass filtering all channels below 38 Hz and assigning the measurements from channel 12 a very high uncertainty (so they will be ignored) and neither method seemed to help convergence, but that is probably a problem with the model itself and not sensor 12.

Conclusions for the GN algorithm

It was decided that the Gauss-Newton algorithm was not the best one for our purposes. It works well for a small number of DOF, but is not be sensitive enough to correct localized disturbances in large DOF structures, which is really what we are trying to do. Also, it does not operate easily on acceleration data and has no way of compensating for measurement noise or model inadequacies. In addition, convergence is sensitive to many variables – among them the Kalman gain, the limits on the search direction magnitude and parameter values, the robustification parameter, iteration tolerance, and the numerical differentiation step size. I decided at this point that it was time to start with a clean sheet and build a real Kalman filter based identification program.

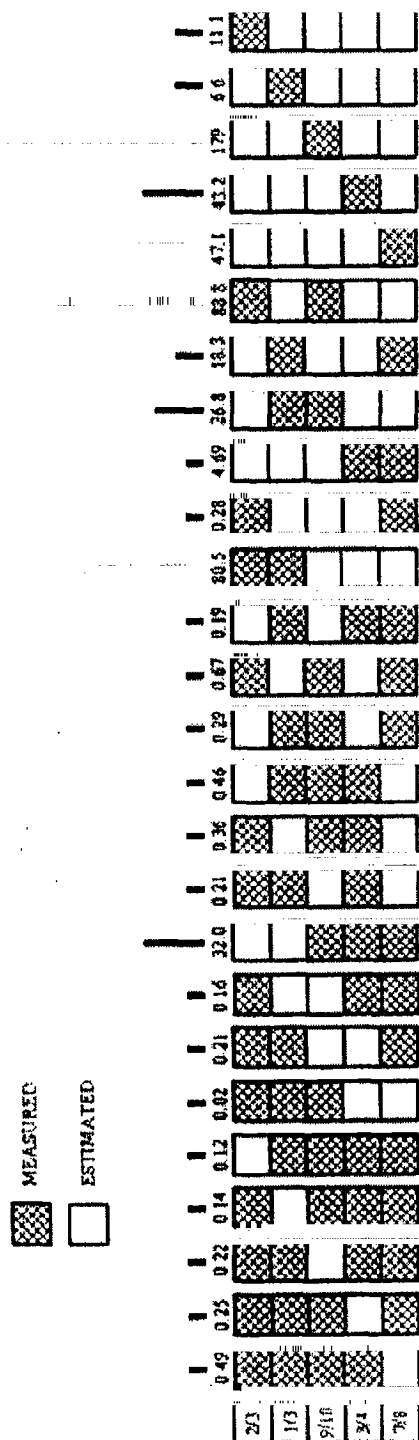


Figure 3.1. Mean error for various states of measurement. Measured nodes are denoted by cross-hatching. The mean identification error is represented by the size of the red bar.

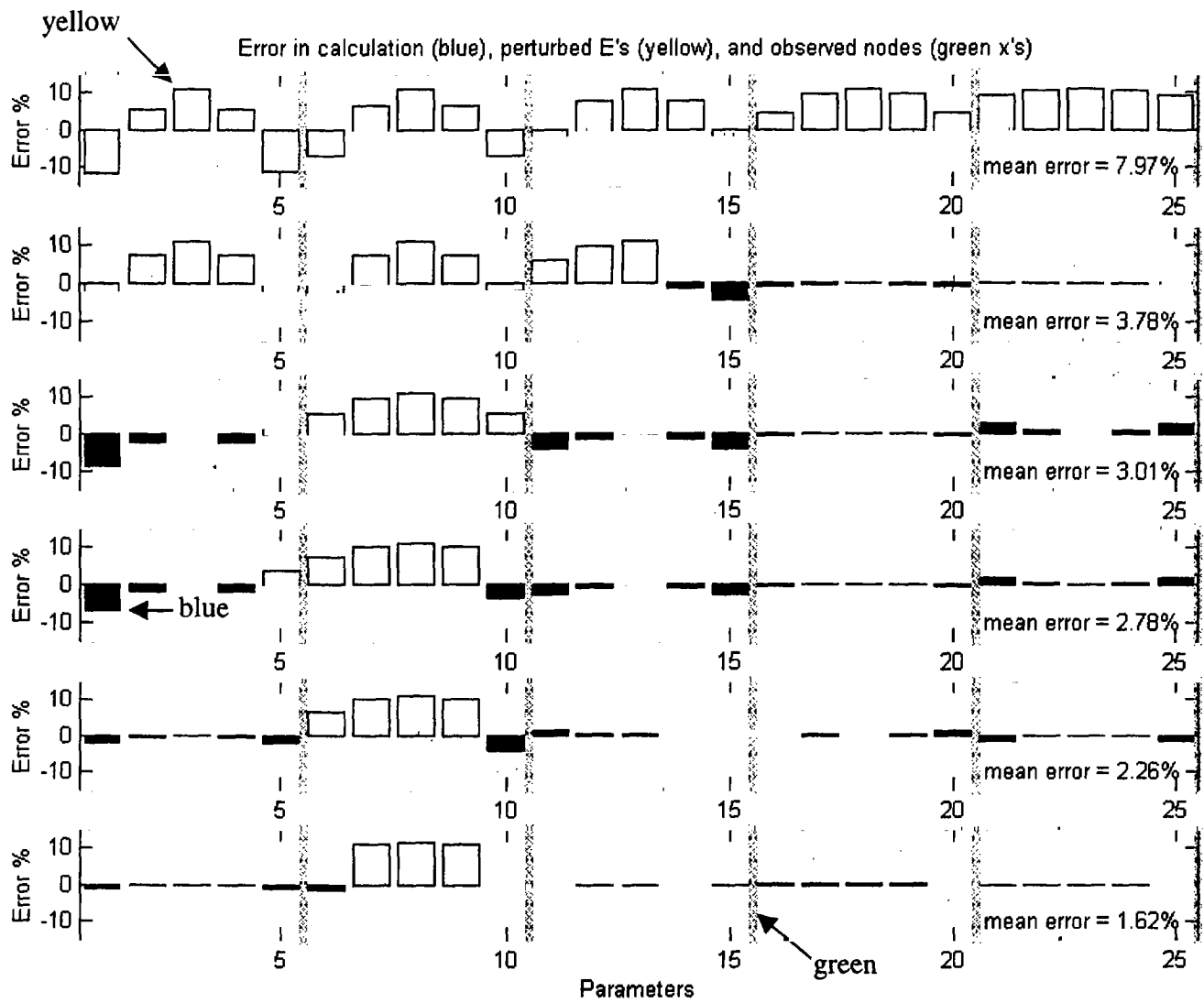


Figure 3.3. Calculation errors for the 50 DOF problem with 5 observations. The errors are shown in the bar graph with the yellow bars denoting elements that were perturbed to 90% of their nominal value and blue bars denoting unperturbed elements. The green lines indicate where the measurements took place.

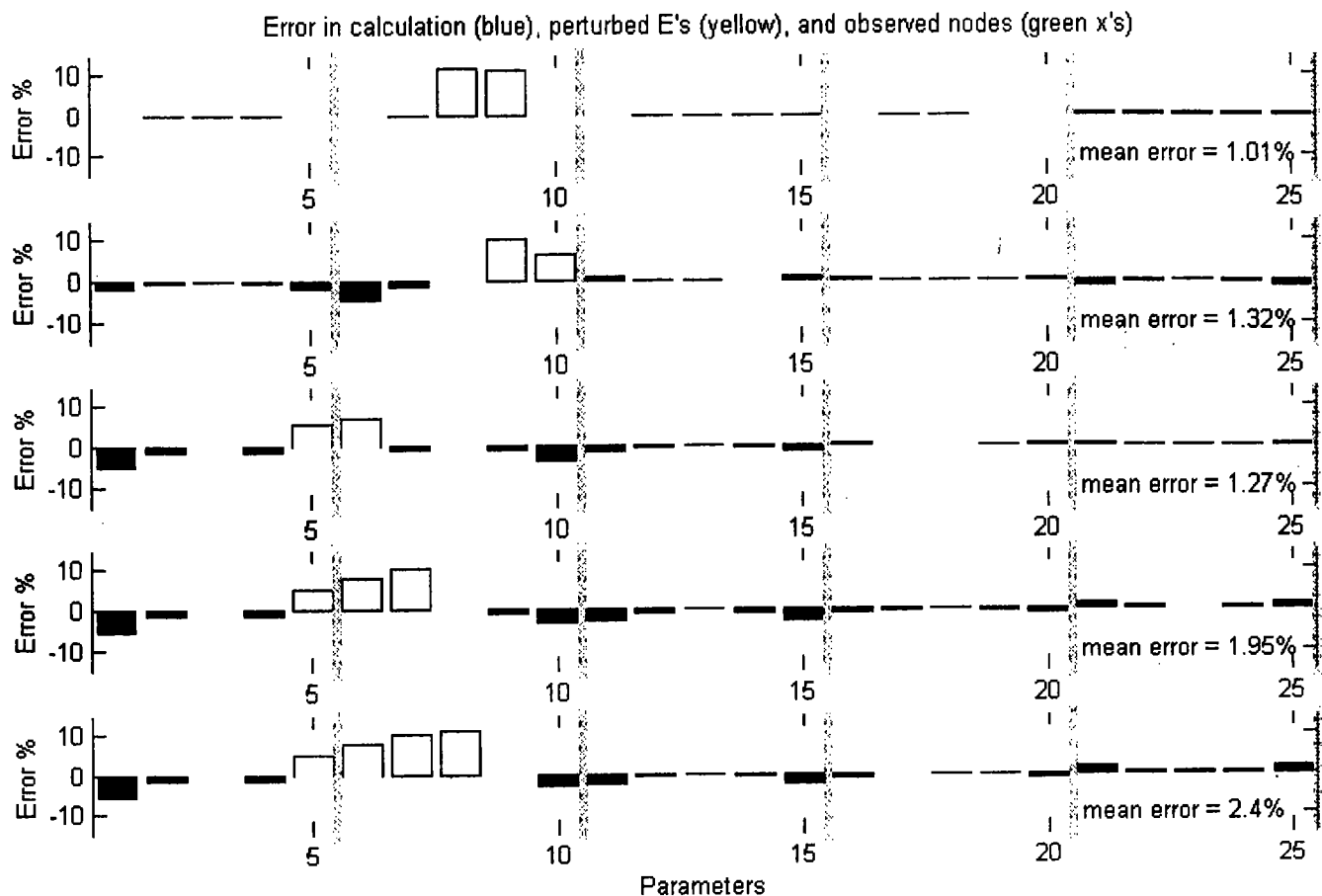


Figure 3.4. Calculation errors for the 50 DOF problem with 5 observations. The errors are shown in the bar graph with the yellow bars denoting elements that were perturbed to 90% of their nominal value and blue bars denoting unperturbed elements. The green lines indicate where the measurements took place.

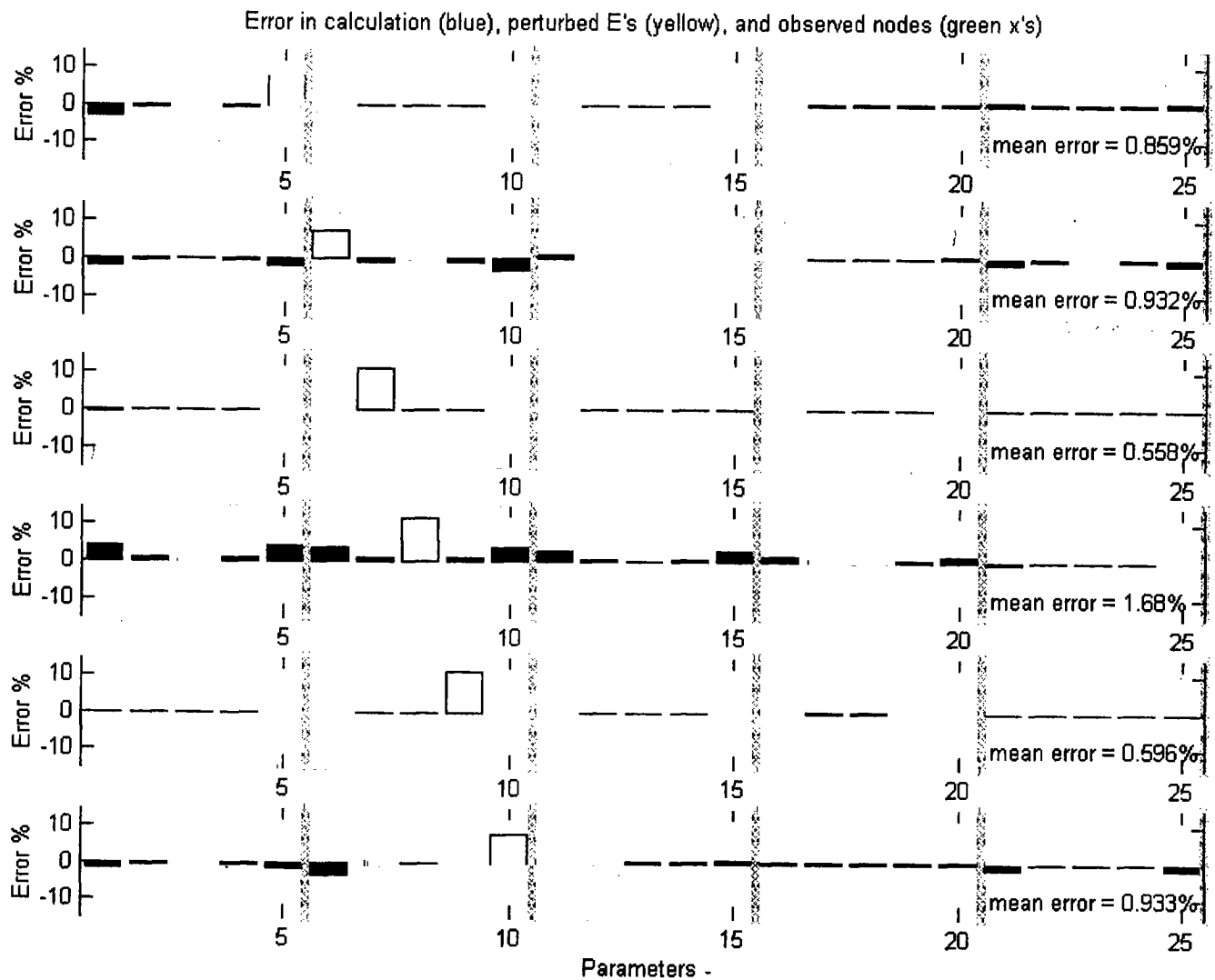


Figure 3.5. Calculation errors for the 50 DOF problem with 5 observations. The errors are shown in the bar graph with the yellow bars denoting elements that were perturbed to 90% of their nominal value and blue bars denoting unperturbed elements. The green lines indicate where the measurements took place.

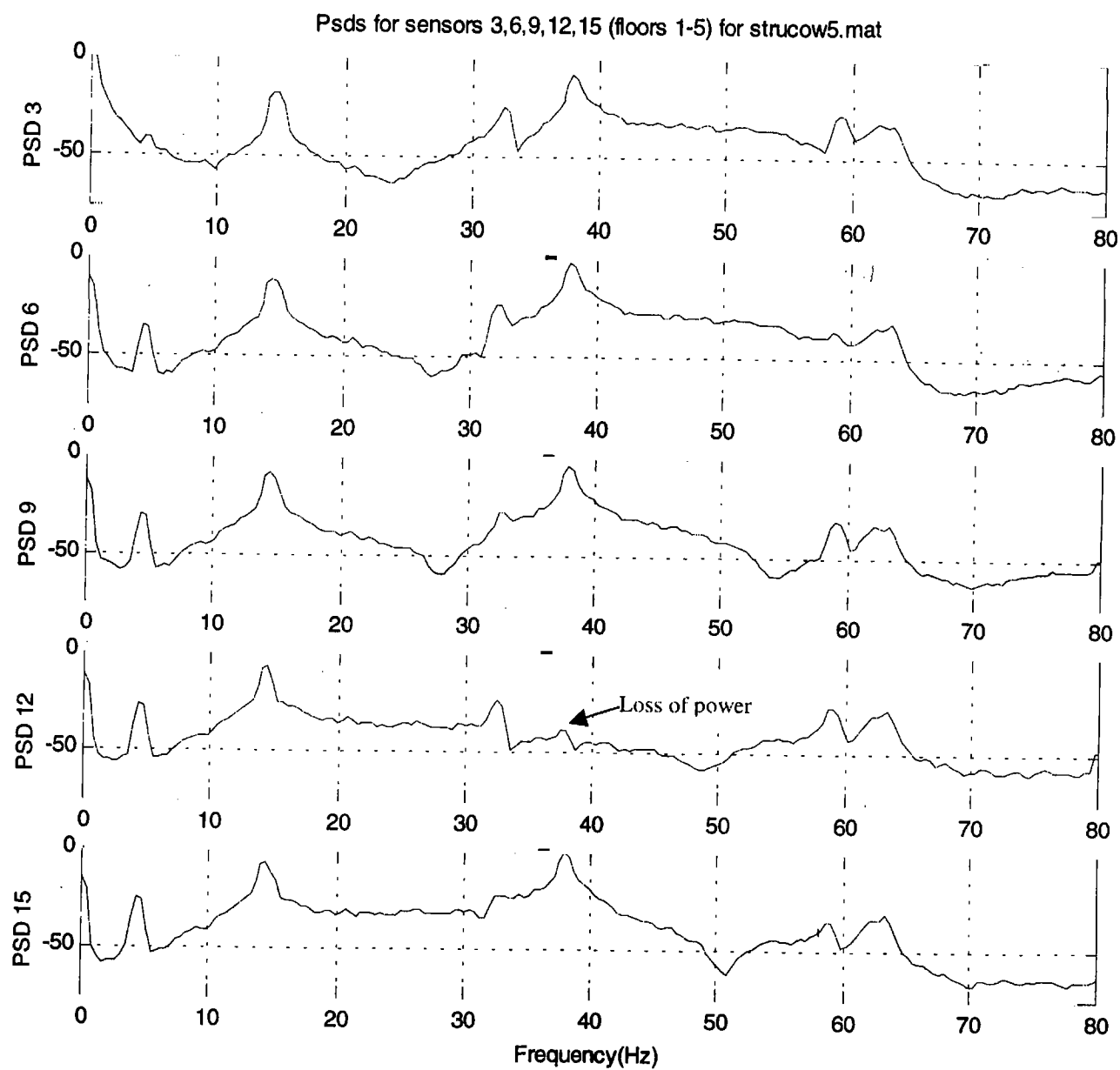


Figure 3.6. Power spectral densities of sensors 3, 6, 9, 12, and 15. Note the lack of power above 35 Hz for sensor 12.

Chapter 4. EKF results

The Extended Kalman filter was covered extensively in Chapter 1, and Gelb is an excellent reference for those more interested in the details. As for results, I will begin with a comparison of the performance of the GN and EKF algorithms for the 10 DOF simulated systems with no noise.

4.1 Simulated models

10 DOF, no noise

To facilitate comparison with the GN iterative algorithm, I ran the same battery of tests on both the EKF and the GN and plotted them on the same plot. For the EKF, I used the following values for the noise parameters (as outlined in Appendix 7.15, stored in `EKF_getnoise.m`):

$R = 0.01 (\pm 0.01 \text{ g's})$ for the measured nodes, 1×10^{12} for the non-measured nodes

$Q_d = P_d = (0.001)^2 = (10\% \text{ of max displacement})^2$

$Q_v = P_v = (0.005)^2 = (10\% \text{ of max velocity})^2$

$Q_k = P_k = (3.1 \times 10^7)^2 = (10\text{x stiffness estimate})^2$

I analyzed 500 samples with a sampling rate of 100 Hz. The driving function was a 1 Hz 1000 amplitude sine wave applied at the first floor. The analyses took about 1.8 hours each on a P3-450 and 2.2 hours on a P2-350.

The results are displayed in Figures 4.1. The blue (right, top numbers) bars are the EKF mean ID errors and the red (left, bottom numbers) bars the GN. It is clear that the EKF works much better overall, with the largest error only 15.6% as compared to 179% for the GN. The GN has better precision on the ones it does well, but it must be realized that the EKF is iterative and will likely beat out the GN if given more samples to operate on. Therefore it is fair to say that the EKF outperforms the GN algorithm across the board with largely equivalent computational times. The EKF has the added advantage of being able to use acceleration data directly and is able to operate in real time as data becomes available. It also seems to have a wider “zone of influence”, being able to converge relatively well with only a single measurement. For the single measurements the following was observed:

Measured node	Error element 1 (%)	Error element 2 (%)	Error element 3 (%)	Error element 4 (%)	Error element 5 (%)	Mean error (%)
Original estimate	-14.5	-33.3	-11.1	-200	-50	61.7
1	4.4	17.9	-1.4	-12.9	-16.7	10.7
2	6.2	-24.6	13.0	-13.3	4.9	12.4
3	2.5	-24.6	11.5	1.8	-34.5	15.0
4	1.9	-23.9	31.4	-6.3	-14.8	15.6
5	-5.8	-22.1	11.2	10.4	17.9	13.5

Table 4.1. The estimation errors of the original estimate and the EKF algorithm results for a single measurement and 500 samples. The errors in bold are the errors for the elements next to the node being observed, the ones in blue are the lowest errors for that particular case and the errors in red are the largest. Note that there is no observed correlation between an observation and the lowest error.

Keep in mind that for the stick model node (or equation) n is located between elements n and $n+1$, so that an observation of node 1 will yield information on elements 1 and 2 while an observation of only node 5 will only directly observe element 5.

The interesting thing is that the mean errors are all about the same after 500 samples. The algorithm is not terribly sensitive to the location of the largest model mismatches. This is in direct contrast to the GN algorithm in which the largest error was when only node 3 (the most closely modeled node) was observed. The error in that case was 179%, and there was no chance of it getting any lower. The error of the EKF when node 3 was observed was only 15.0%, and with more samples it is possible that the error will decrease. The sensitivity of the GN algorithm to initial model accuracy is a major shortcoming.

Another interesting thing was the lack of correlation between measurement and low error rates. Indeed, there is only a single occurrence (node 3 measured) of an element by a measured node having the lowest error. It seems for this simple system that for the EKF the point of measurement is not a critical parameter, unlike the GN algorithm.

One last consideration: The input function of this experiment was a sine wave, the convergence may be more rapid and complete with a white noise source (see Appendix 7.20 for

information on `idinput.m`). I did not have time to rerun this experiment with a pseudorandom source, but the difference in convergence time should be similar for both algorithms, so the relative differences should be about the same.

10 DOF with noise

Here, the 10 DOF (5 translational and 5 rotational) model was tested with different levels of noise and with a variety of measured and unmeasured nodes. The rotational nodes were not considered measured, and the number of translational nodes measured was varied. The unmeasured nodes were assigned a variance of 1×10^{12} , effectively rendering each “measurement” completely ineffective in updating the state. With no added noise, the EKF can identify the system quite well for just about any combination of measurements, including only 1 or 2 measurements. It is much more precise and robust than the Gauss-Newton iterative search.

Experimental setup

To continue the examination of the performance of the EKF for the 10 DOF system, white noise with different S/N ratios was added to the simulated measurements and different input functions and values for the measurement covariance matrix **R** were used. The white noise was added at levels of -20 and -10 dB S/N. For example, for a S/N of -20 dB, white noise with a maximum amplitude of 1/10 of the maximum of the corresponding clean measurement was added to each measurement. The noise was therefore a different level for each node, as it depended on the level of the measured acceleration of that node. A noise level of -20 dB is a significant amount of noise for sensitive accelerometers and should be a good indication of noise robustness. The measurement noise covariance was approximated by a constant for early experiments; in later trials the actual covariance of the noise added to the measurements was used.

There are several degrees of freedom here, namely:

- I. The type of excitation (white noise, sine wave)
- II. The level of noise added to the signal (none, -20 dB, -10 dB)
- III. The size and composition of the **R** (measurement covariance) matrix
 - a. Same for all nodes (0.01, 0.1)
 - b. Different for each node (measured for each one)
- IV. Which nodes are observed (all, some, one)

I therefore ran several trials, varying one parameter at a time to come up with the best performance. My standard perturbation of the initial parameter estimate * [7/8 3/4 9/10 1/3 2/3] will be used. That is, the actual structure will have k values that are 7/8, 3/4, 9/10, 1/3, and 2/3 of the estimated k values. The experiments were performed in the following order:

1. White noise input, no noise added, $\mathbf{R} = \text{diag}(0.01)$ (lower bound)
2. White noise input, no noise added, $\mathbf{R} = \text{diag}(0.1)$ (upper bound)
3. White noise input, no noise added, $\mathbf{R} = \text{measured}$ for each node

Of these experiments, trial 2 yielded the best results. Therefore I have fixed \mathbf{R} at $\text{diag}(0.1)$ for the rest of the experiments. It is not as good for some no-noise situations, but it is essential to only change one variable at a time. The other two experiments with white noise inputs were

4. White noise input, noise added at -20 dB, $\mathbf{R} = \text{diag}(0.1)$
5. White noise input, noise added at -10 dB, $\mathbf{R} = \text{diag}(0.1)$

This should give us a good idea of the ability of the algorithm to converge in the presence of noise.

Incidentally, when using a white noise input the performance of the algorithm would fluctuate, sometimes significantly, so some of the above results are averages over several experiments. These fluctuations are not present when a sine wave input is used. To fix this, a pseudorandom input was generated using `idinput.m` (see Appendix 7.20). The results were much more uniform when this input was used, but this experiment was not repeated using the pseudorandom input due to a lack of time.

I now repeat the above using a sine wave input, with frequency 1 Hz.

6. Sine wave input, no noise added, $\mathbf{R} = \text{diag}(0.01)$ (lower bound)
7. Sine wave input, no noise added, $\mathbf{R} = \text{diag}(0.1)$ (upper bound)
8. Sine wave input, no noise added, $\mathbf{R} = \text{measured}$ for each node
9. Sine wave input, noise added at -20 dB, $\mathbf{R} = \text{diag}(0.1)$
10. Sine wave input, noise added at -10 dB, $\mathbf{R} = \text{diag}(0.1)$

Again, using $\mathbf{R} = \text{diag}(0.1)$ led to the lowest error rates. The results for two separate iterations are shown in Figures 4.2 and 4.3. The mean identification errors when the last 100 estimates are averaged are shown in blue, and the single last estimate is shown in red. This can

help us see where there is still some oscillation around the correct answer. It is clear from the differences in Figures 4.2 and 4.3 that the accuracy of the identification can vary significantly if the white noise (random) input is used. The sine wave input, on the other hand, yields identical answers when there is no added white noise (as expected) and exhibits only a slight variation

when white noise is added to the measurements. As it returned the more repeatable results, for the observability test the sine wave input was used, as I had not yet been directed to `idinput.m`.

Conclusion for 10 DOF

The EKF works quite well on the 10 DOF system, with and without noise present. With – 20 dB of noise and a sinusoidal input, the mean identification error rate was only a few percent. The next test should be combining a low number of measurements with added noise to see if identification is still possible with only one or two sensors in the presence of noise.

50 DOF without noise, only floors measured

This model is the same as the 10 DOF with the columns discretized into 5 sub-columns. The measurements are assumed to take place only at the floors, so the number of measurements stays the same at 5 but the number of DOF goes up by a factor of 5 to 50.

I have not yet been able to make any progress on the 50 DOF model of the five-story building. I have varied the noise covariance matrices, the input amplitude, the size of the perturbation, and the location and size of the perturbations, and I can get no convergence at all. Worse, the algorithm is very slow – about 50 hours for 100 samples. This is due to the very large size of P : 125×125 . This means there are 7750 independent members of P that have to be propagated, so `ode45` has to be run 7750 times per time sample. This is quite costly, and we shall examine methods to reduce that cost at the end of this chapter.

However, even with the high cost we still have no positive results. The nodes that are perturbed are not detected, even with changes of up to 50%, and nodes far away from the damage are occasionally incorrectly identified with errors ranging up to 50%. With the same amplitude input function as used for the 10 DOF, the changes in the output due to the perturbations (even for 20% changes in stiffness) were very small. It was necessary to multiply the input by a factor of 1000 in order to make the errors large enough to affect the output, something that should not be necessary as the 10 and 50 DOF systems are very similar models.

The estimation results for 5 of the 25 of the stiffness values (these are ones located next to the floor nodes) for 100 samples are shown in Figures 4.5. It is clear that there is little convergence, except possibly for states 110 and 115. The other three estimates are very poor and are headed in the wrong direction. It is possible that a larger number of samples would yield better results, but as it currently takes an hour to compute two samples, further computations have not yet been attempted. It is also possible that we are suffering the same problems as before using a white noise input, but I have seen the same results five times in a row, making the possibility remote. To be sure, I re-ran the experiment with a sine wave input and got the same results, so the problem is not in the input.

I have an idea as to why identification of this system has been difficult. This is a model that can be completely specified by 10 DOF, yet we are attempting to describe it using 50 DOF. Thus we have the classic problem of over-specification, where many of the DOF are not necessary to describe the system. This may be the reason we are having trouble identifying it. It is also true that over-specification is the exact opposite of the situation we will commonly encounter in practice, where our model will most likely under-specify the system. Thus it may not be the best test of the algorithm. I would suggest a different model, one that requires 50 or more DOF in order to completely describe the system. This may be the only way to effectively test the algorithm. We will still be limited by the long computational times, but at least we will be able to more effectively test the algorithm.

To test my hypothesis, I used the SVD of the generalized Hankel matrix to estimate the order of the matrices for the 10, 50, and NTS 5 DOF systems (see Appendix 7.17). The results confirmed that only 10-14 DOF are needed for the 50 DOF system. I have also run some experiments in which the 10 DOF model was successfully used to identify the 50 DOF simulated system (see Appendix 7.20), indicating that most of the DOF of the 50 DOF model are superfluous and are preventing convergence.

10 DOF using all K_{ij} values

It was suggested by Dave that I use, as parameters, all of the values of \mathbf{K} instead of the underlying independent stiffness values. Not every K_{ij} is used as \mathbf{K} is symmetric, so only the diagonal and top triangle are used. This would give the program maximum flexibility in determining the changes in \mathbf{K} that were necessary to match the model. This was to help with the

NTS processing, but I wanted to try it on the 10 DOF system first to determine the effect of having more parameters than DOF. This procedure lengthened the computational time considerably as the number of parameters increased from 5 to 27, which increased the number of states from 25 to 47. A perturbation of 50% was applied to all the parameter estimates (the real K was $\frac{1}{2}$ of the estimated K). Thus the initial error for all the nodes was 100% ((actual – estimate) / actual).

The results are shown in Table 4.2 at the end of this Chapter. Interestingly, the algorithm did converge to the correct eigenfrequencies (second part of Table 1) even though it only 9 parameters of the 27 converged. These parameters, though, were ones that were stiffness parameters or sums of stiffness parameters. The parameters that didn't converge are those in which there was a rotational component added in. The uncertainty for the rotational components is essentially infinite, so anything with one of those factors doesn't converge.

4.2 NTS 5 DOF

This is a 5 DOF model that describes the large 5-story model instrumented at NTS. I believe this model has the opposite problem as the one described in the 50 DOF section, as it is far too low order to describe the system effectively. The SVD of the Hankel matrix (see Appendix 7.17) confirmed that the system is under-specified. I have tried many different values for the noise matrices P , Q and R , and have not been able to get the system to converge, even for up to 59 seconds of data at 400 samples/second. The stiffness values do not change much at all or just continue to change very slowly, with no hint of convergence. At Dave McCallen's suggestion, I even tried changing the algorithm so that every independent value of K would become a parameter, in order to ensure that the algorithm had the most flexibility possible. This worked on the 10 DOF discussed above, but not on this system.

I have proposed to Dave McCallen that we first try to identify a simpler known system, such as coupled oscillators with three or more masses. This would allow us to test the algorithm with real data, yet the system would be well understood and easily described with only a few DOF. I would also like to be able to add a fourth, smaller mass to the system to explore to what extent model inconsistencies may be accounted for in the designation of Q , the system propagation noise matrix. The nature of the EKF is that some modeling inaccuracies may be compensated for by enlarging Q , but how large the inaccuracies may be and the amount of Q

change required is not clear. It would be nice to be able to describe this quantitatively. This experiment would also make a good paper.

Conclusions for the EKF

I have been able to show that the EKF works quite well with the 10 DOF simulated system, even with -20 dB added noise in the measurements. However, this is a simulated system, and the modeled and actual systems are the same order – indeed, they are identical except for the stiffness values. It may be interesting to change the experiment slightly so that the model is only 8 DOF, and then see if we can still estimate the system well. It may also be useful to process the measurement data to get an idea of what order model should be used to simulate the system. There are several ways to do this, using singular value decomposition and others, and it may be useful to look into that. However, for this year, we may want to limit ourselves to understanding the limits under which the EKF and its like may operate successfully.

I also believe that the 50 DOF model should be revamped to represent a system that requires close to 50 DOF to be described correctly. As for the NTS structure, I think we should start a little smaller and instrument a real low DOF system that we can describe using a simple model. That way we can wring out the algorithms and the noise problems more systematically.

4.3 Speeding it up

Finally, a note about the speed of the program. Right now, for the 10 DOF system, the program takes about 6000-8000 seconds to run for 500 samples, which is about 12-16 seconds per sample. Not exactly real time, but do-able. The steps that are taking the longest are the solving of the continuous differential equations for the \mathbf{P} calculation between measurement points.

The process begins by peeling off all of the diagonal and upper triangular values and putting them into vector form using `mat2vecs.m`. Then `ode15s.m` is called to do the ODE calculations.

The vector is at most $\sum_1^{Mord} k$ members long, where $Mord$ is the model order, and can therefore

take a while to compute. I have experimented with different methods that would not require \mathbf{P} to be calculated at each time step. This included the simplistic arrangement in which \mathbf{P} is calculated every 10th time after the first 20 samples, and another in which \mathbf{P} is only calculated if

the change in the estimated stiffness parameters changes by more than a threshold percentage.

The results were not terribly convincing, as shown in Table 4.3.

Trial	Time (hours)	Convergence time (samples)	P threshold (%)	Mean error (%)
1	4.6	50	0	0.4
2	1.96	~80	2	7.1
3	1.70	~50	5	4.5
4	1.75	~60	10	4.5
5	1.79	~60	20	4.5
6	2.2	N/A	Every 10 th sample	232

Table 4.3. Data concerning the use of an adaptive (2-5) and constant (6) method to restrict the number of times \mathbf{P} is calculated.

For thresholds of about 5-10%, the time is cut in half but accuracy suffers, on the order of about 5%. Interestingly, the largest error for trials 2-5 was for element 5, all the other elements were still identified to within around 1% error. It may be useful to use a threshold of about 5-10% if great accuracy is not needed.

However, following suggestions from the Big Four (Dave McCallen, Dave Harris, Jim Candy, and Larry Ng) I made some additional attempts to speed up the program. These are detailed in Appendix 7.20, the results of which are reproduced here.

I again concentrated on approximating the $\dot{\mathbf{P}}$ calculation, as that is where the most computational expense is located. These include using a linear and quadratic interpolation for $\dot{\mathbf{P}}$, using a constant \mathbf{P} after $\Delta\mathbf{P}$ dropped below a threshold (above), and compiling parts of the $\dot{\mathbf{P}}$ calculation. I tried four techniques this time. In the first I looked at the average change in percent of \mathbf{P} from one sample to the next. This was to examine \mathbf{P} to see if it converged to a steady state value and could therefore be expressed as a constant after some time. I knew it did not converge, but I wanted to know why so I plotted the changes in \mathbf{P} vs. time.

The result is shown in Figure 4.5. It is clear that the members of \mathbf{P} do not converge to some steady-state value; there is still significant change after 75 samples, the convergence time. Thus it will not be possible to use a constant \mathbf{P} after thresholding to save computational cycles.

The second technique considered was to take the difference of *each member* of \mathbf{P} just before and just after the update calculation. If the difference was below a threshold, *that member*

of \mathbf{P} was held constant and not propagated using ODE45.M, saving comp cycles. This differs from the technique above, in which the entire \mathbf{P} was held constant if the mean change was below a threshold. Each member was tested each time, so that a member could be held constant one round and then be propagated the next. This resulted in significant computational overhead, but I wanted to see if there was a threshold we could use that would save enough time to justify the extra computational expense.

The results from the second test are more involved and are tabulated in the first 11 rows of Table 4.4. Some experiments were run more than once to get an idea of the variability of the results. The criterion for convergence was a threshold in the change in stiffness states, detailed above. It is clear that a $\Delta\mathbf{P}_i$ threshold of about 1% was the maximum that could be used without raising the error rate to unacceptable levels. Unfortunately, this threshold did not result in a reduction in computation time, even though almost 5% of the ODE calculations were skipped. It is thought that the overhead associated with keeping track of the changes in \mathbf{P} probably results in little computational savings. Also, the reduced accuracy due to the approximation in \mathbf{P} means that the algorithm does not converge as quickly to the correct values, somewhat increasing the calculation time. Therefore this option does not seem to be a wise one – approximating changes in \mathbf{P} results does not significantly reduce computational expense.

The third method was to limit the area of \mathbf{P} that was calculated each time. Since the stiffness states are the only ones of interest, it might be that the \mathbf{P} calculation is only critical where the stiffness states are concerned. Perhaps it would be possible to set the \mathbf{P} calc constant for the displacement and velocity parts and only calculate those parts of \mathbf{P} related to the stiffness states. The result is shown in row 12 of Table 4.4. It actually seems to work pretty well, enabling a reduction in calls to ODE45.M by over 95% (reducing the runtime by over 50%) and still getting pretty good stiffness estimates (8.2% error) by averaging the last two hundred points. By plotting the estimates, it was clear that they oscillated about the actual stiffness values, indicating weak convergence. This warranted further exploration. A compromise was tried in which all of \mathbf{P} is calculated every 5th or 10th step and the stiffness state propagation is calculated the rest of the time (every sample). The results are shown in Table 4.4 in rows 13 and 14. It worked surprisingly well, achieving a mean identification error of only 1.16% (1.39% with the last two hundred estimates averaged) for every 5th calculation to 3.35% (1.97%) for every 10th.

At the same time, it took only slightly longer than the threshold of 1% and about $\frac{1}{2}$ the time of the 5% ΔP_i threshold.

The fourth method is quite simple – just calculate \dot{P} every 5th or 10th time and consider it constant the rest of the time, skipping the stiffness updates. This did not work before (see above), but for some reason worked this time. I think I fixed a bug in-between the test above and the new tests, but for whatever reason it now works quite well, with 80% of the ODE45.M calculations skipped and identification error rates of only 0.39% calculating every 5th \dot{P} and 90% skipped with an error rate of 1.97% for every 10th. This accuracy held up for a variety of inputs. It seems the stiffness states need not be calculated each time to maintain a decent level of accuracy. As the run times were essentially the same, calculating \dot{P} every 5th sample seems to be a good choice. I think this is the best way to achieve reasonable identification accuracy while saving some computational expense. Of course, if everything went perfectly it would still be faster to calculate \dot{P} every time, as the convergence is faster. But for situations where convergence is not rapid or assured, only calculating \dot{P} every 5th step may make sense.

Larry has some further ideas about approximating P and K , and hopefully these will speed up the processing even more. If not, though, this is the perfect problem to use parallel processing as the ODE calculation is performed on each of the members of \dot{P} , none of which depend on the others for calculation. Therefore the program could run normally until the ODE calculation is required which could then be run in parallel, greatly speeding the calculation process.

```

10 DOF, all measured
Nsamps = 1000;
perturb = 0.5;
all_k = 1;
EKF_test_parsdiff_10DOF_all_k.m
10 DOF all_k.txt

```

(i,j)	pars	FE model	true struct	est. model	%diff

		1e6.*			
1,1	1	3.2443	1.6222	1.6216	0.035276
2,2	2	9.7697e+007	4.8849e+007	9.7697e+007	-100
1,3	3	-1.6222	-0.81108	-0.80802	0.37755
2,3	4	116.8	58.398	116.8	-100
3,3	5	3.2443	1.6222	1.6059	1.0006
1,4	6	-116.8	-58.398	-116.8	-100
2,4	7	5606.2	2803.1	5606.2	-100
4,4	8	9.7697e+007	4.8849e+007	9.7697e+007	-100
3,5	9	-1.6222	-0.81108	-0.81142	-0.04138
4,5	10	116.8	58.398	116.8	-100
5,5	11	3.2443	1.6222	1.6387	-1.0184
3,6	12	-116.8	-58.398	-116.8	-100
4,6	13	5606.2	2803.1	5606.2	-100
6,6	14	9.7697e+007	4.8849e+007	9.7697e+007	-100
5,7	15	-1.6222	-0.81108	-0.81913	-0.99199
6,7	16	116.8	58.398	116.8	-100
7,7	17	3.2443	1.6222	1.8753	-15.606
5,8	18	-116.8	-58.398	-116.8	-100
6,8	19	5606.2	2803.1	5606.2	-100
8,8	20	9.7697e+007	4.8849e+007	9.7697e+007	-100
7,9	21	-1.6222	-0.81108	-1.0098	-24.501
8,9	22	116.8	58.398	116.8	-100
9,9	23	1.6222	0.81108	0.96963	-19.548
7,10	24	-116.8	-58.398	-116.8	-100
8,10	25	5606.2	2803.1	5606.2	-100
9,10	26	116.8	58.398	116.8	-100
10,10	27	9.7686e+007	4.8843e+007	9.7686e+007	-100

Average E error = 69.0044 %.

FE e-freq	Est. e-freq	Actual e-freq	%diff

0.84953	0.60238	0.60071	0.27843
2.4682	1.7554	1.7453	0.58252
3.8554	2.8027	2.7262	2.8065
4.8907	3.5806	3.4583	3.538
5.5081	4.021	3.8948	3.2388

time = 107 minutes

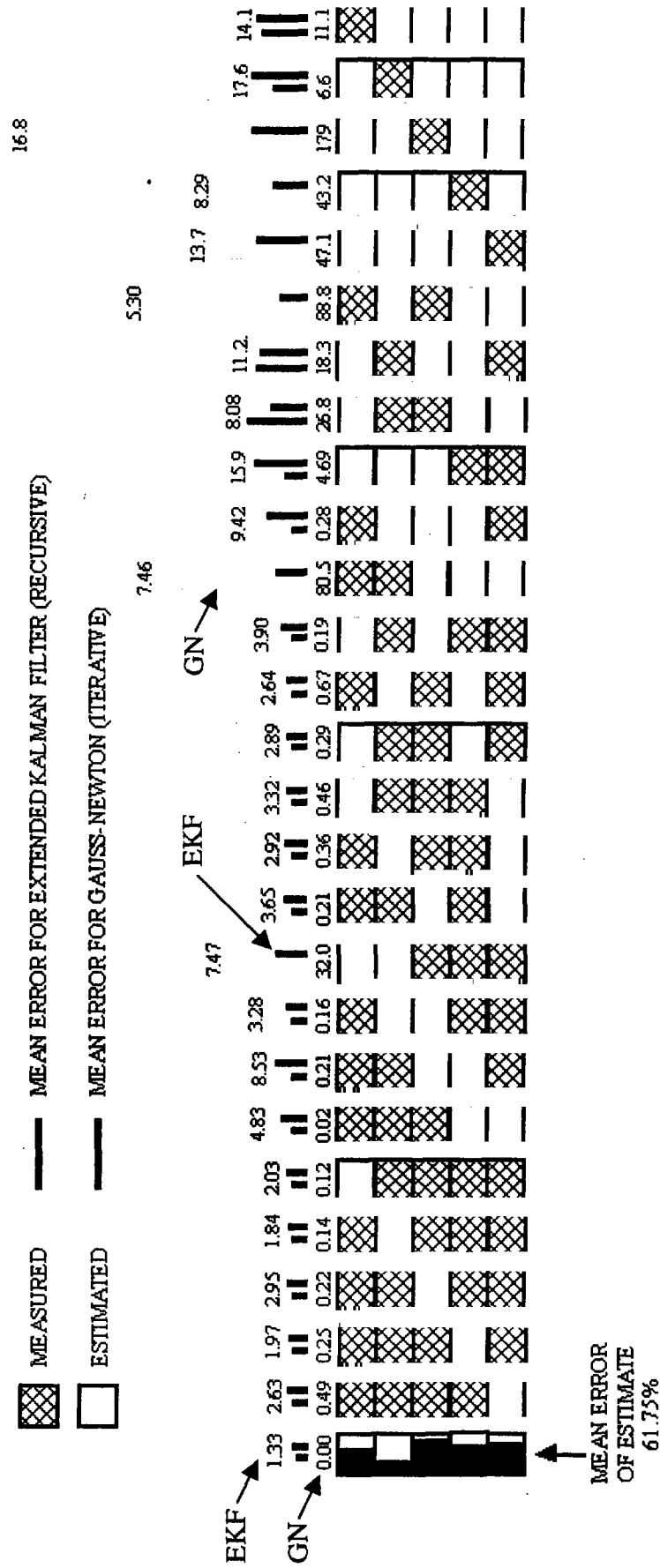
Table 4.2. The results from the 10 DOF experiment where all possible K_{ij} values are used as the parameters. Only the highlighted parameters converged.

Exp #	Threshold (%) or parts calculated	Run time* (min)	Samples to converge	ODE calc skipped (%)	Mean ID error (%)	Max ID error (%)
1	None	5.88	127	0	0.62	1.68
2	None	6.59	127	0	0.62	1.68
3	10^{-16}	7.99	127	0.002	0.63	1.68
4	0.001	7.93	127	0.17	0.66	1.79
5	0.01	7.88	127	0.39	0.65	1.78
6	0.1	8.15	127	1.03	0.38	0.79
7	0.1	7.67	127	1.03	0.62	1.61
8	1.0	8.24	134	4.65	0.30	0.65
9	1.0	6.95	134	4.84	0.35	0.77
10	2.5	19.48	N/A	8.18		
11	5.0	18.54	N/A	15.2		
12	Only stiffness	8.99	N/A	95.4	30.8 (8.20) ¹	77.6
13	Every 5 th with stiffness	10.0	~150	75.4	1.16 (1.39) ¹	-2.84
14	Every 10 th with stiffness	9.96	~150	84.2	3.35 (1.97) ¹	-7.88
15	Every 5 th w/o stiffness	9.62	~100			
16	Every 10 th w/o stiffness	9.77	~150	90.0	1.97 (1.65) ¹	-3.98

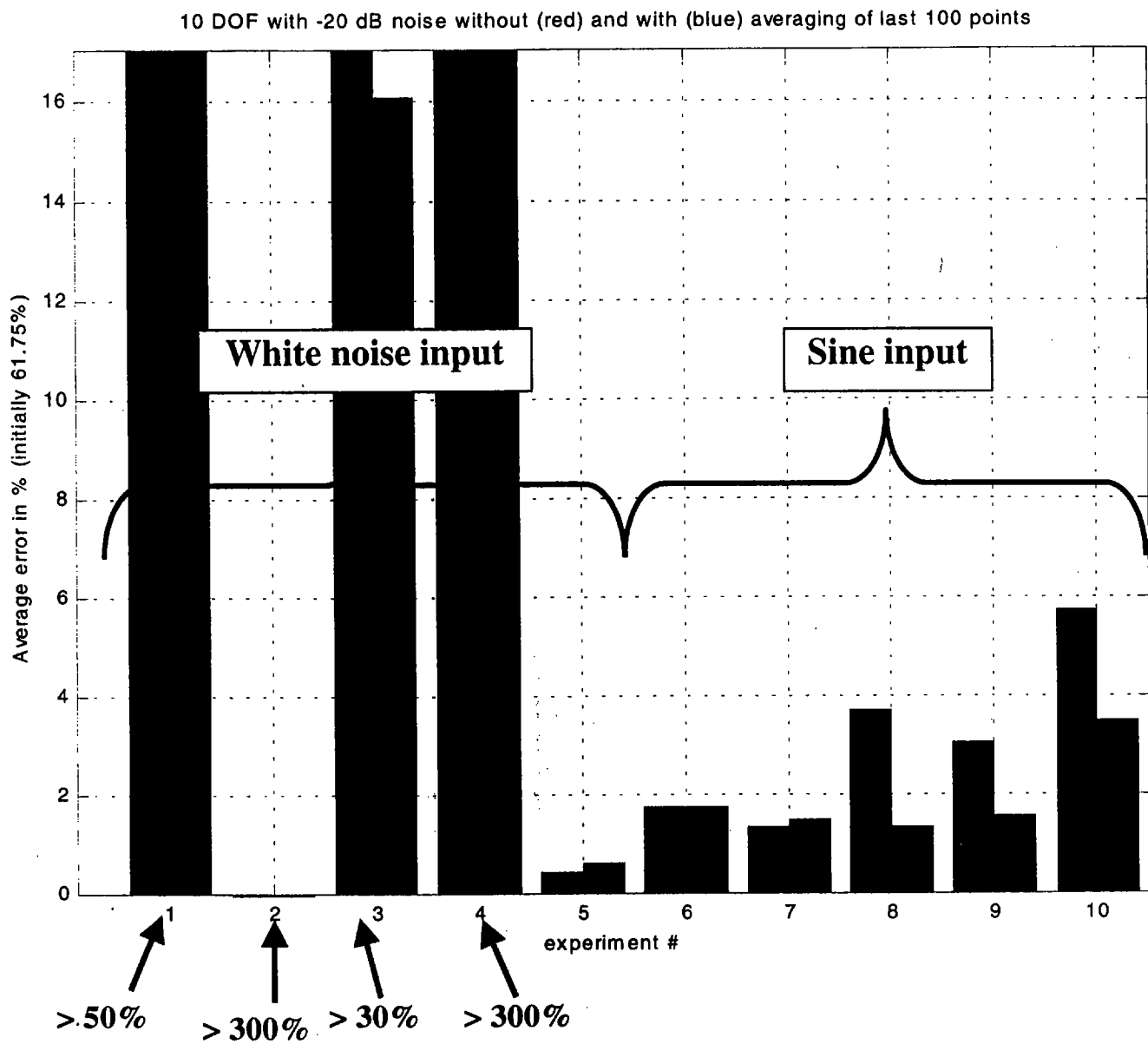
Table 4.4. Convergence times and identification errors for the same input/output sequences and different ΔP thresholds and conditions. Every nth with stiffness means that the Pdot values associated with the stiffness states were calculated at each time sample and the entire Pdot calculation was done every n samples. Every nth without stiffness means that only the entire P calculation was done every n samples.

* run time can vary depending on the amount of memory available, accurate to about +/- 1 minute

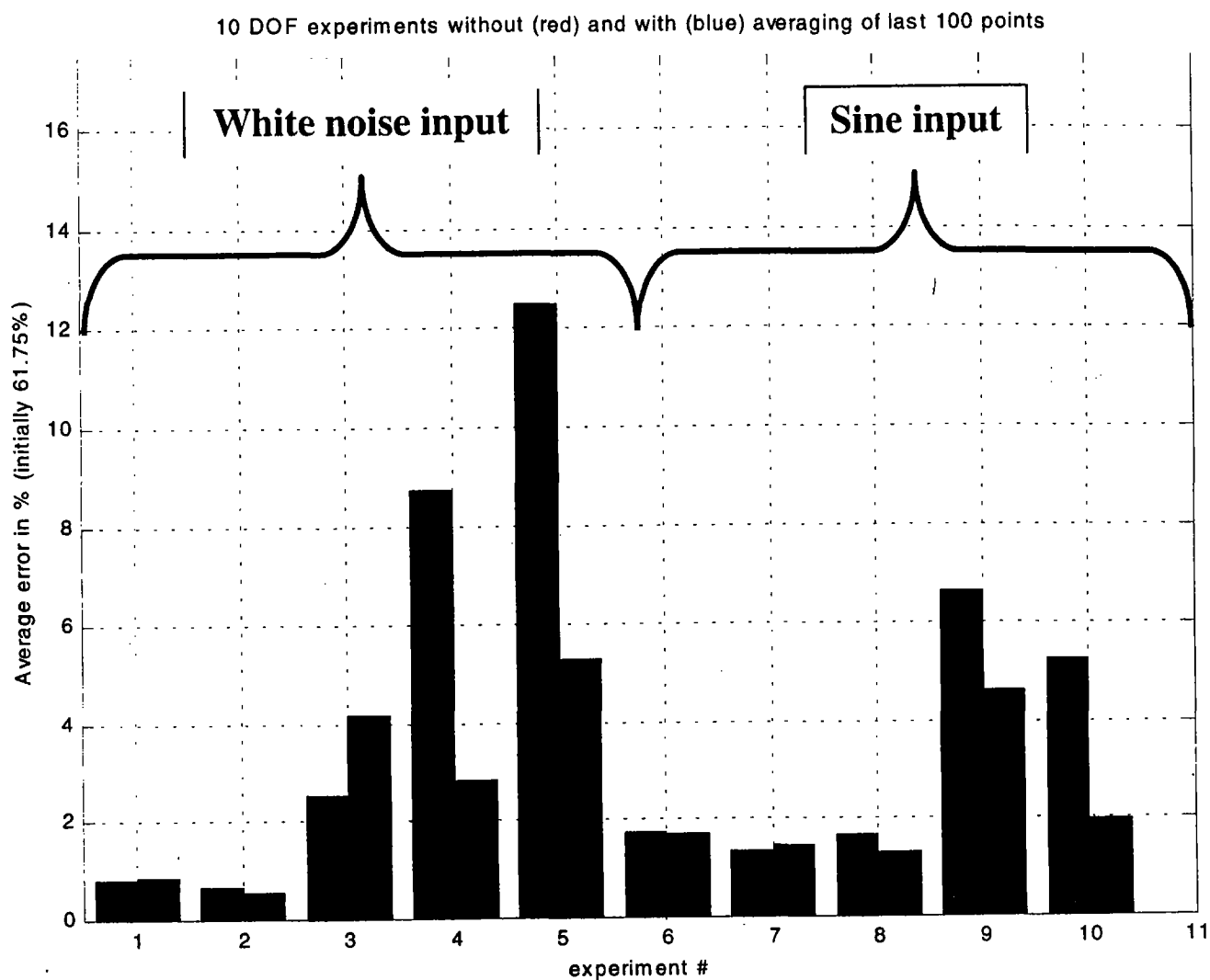
¹ average error using stiffness values defined by the mean of the last two hundred estimates



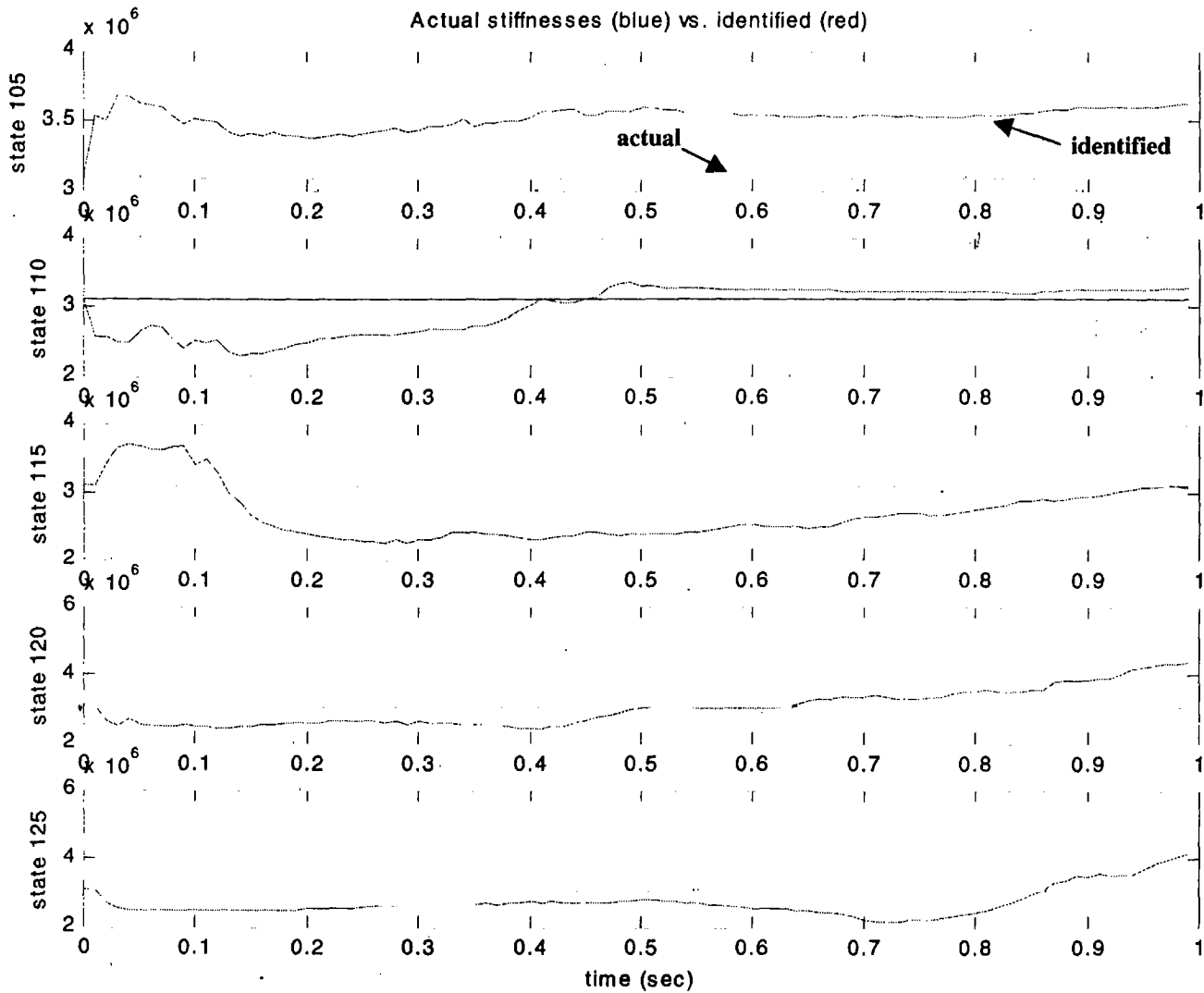
Figures 4.1. Mean identification error for 10 DOF using Gauss-Newton method (red bars on left) and the EKF with sine input, $R = 0.1$, no added noise (blue bars)



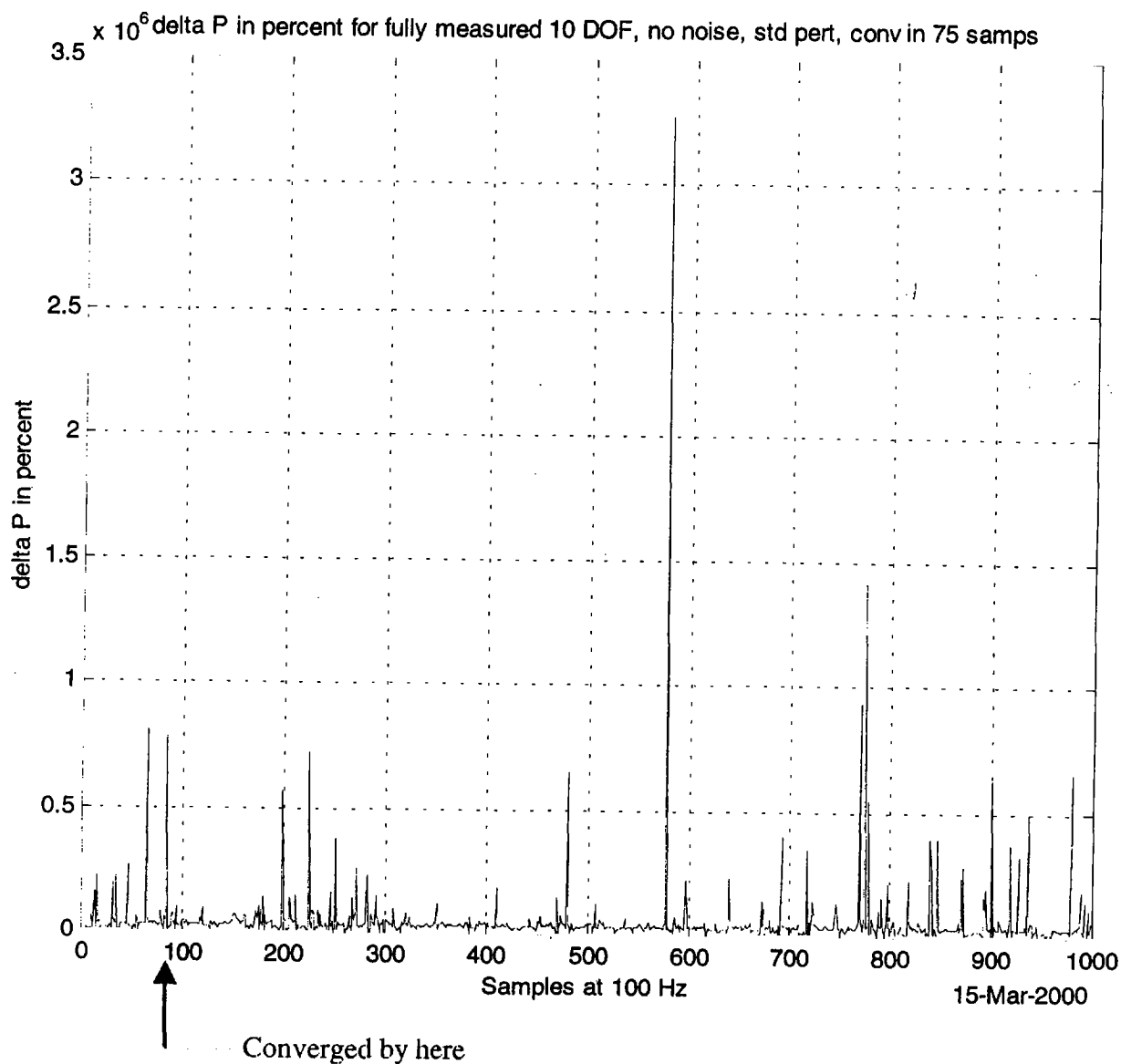
Figures 4.2. Average identification error for the EKF with a -20 dB noise level. This is the first repetition of the 10 experiments above. The left (red) bar represents the mean identification error with no averaging, while the right (blue) bar is the error with the last 100 samples averaged. It is clear that the sine wave input results in a more accurate identification.



Figures 4.3. Average identification error for the EKF with a -20 dB noise level. This is the second repetition of the 10 experiments above. The left (red) bar represents the mean identification error with no averaging, while the right (blue) bar is the error with the last 100 samples averaged. It is clear that the sine wave input results in a more consistent performance.



Figures 4.4. The actual values for five of the 25 stiffness value to be identified (blue), and the estimated values (red). These five values are next to observed nodes and should be easily identified. There were 100 samples analyzed. Note that the y axis is not the same scale for every plot.



Figures 4.5. The change in error covariance P in percent for 1000 samples at 100 samples/second. Note the scale here is multiplied by 10^6 , the maximum value being $3.5 \times 10^6\%$.

Chapter 5. Conclusions

So what have I learned?

The GN algorithm is not a good one for real systems in which the data is acceleration, the numbers of DOF are high, the number of measurements is low, and the model is inadequate. It is simply not robust enough to function well in the environment that we desire.

The EKF shows a lot more promise, as the amount of noise in the system, measurements, and model may be quantified. It is also readily scalable to large DOF, although more work will have to be done to speed it up for high (> 50 DOF) systems. It also does not require full measurement and works easily with acceleration.

Why the 50 DOF model doesn't work

As explained in Chapter 2, the 50 DOF model is just the 10 DOF with the columns divided into 5 sections. I began to suspect the model was flawed (as far as parameter identification goes) when I could not identify the 50 DOF simulated system with either algorithm even with no noise present and all 25 translational nodes measured. Since I know everything about the system, I should have been able to identify it correctly and was not.

I began by examining the expected model order using the singular value decomposition (SVD) of the generalized Hankel matrix (see Appendix 7.17). This method uses the measured output sequences to estimate the model order, and it came up with about 10-14 DOF needed, far short of the 50 DOF used. That means that most of the DOF are not needed and the system is very over-specified. This type of system can cause algorithmic instabilities and can prevent the algorithms from converging.

In addition, as one of the suggestion from the Big Four (Dave McCallen, Dave Harris, Jim Candy, and Larry Ng, see Appendix 7.20), I took the five simulated outputs from the 50 DOF model and used the 10 DOF model to identify the parameters I had changed. It identified them all quite well, indicating that in this case 10 DOF was enough to identify the system. Therefore the 50 DOF model is over-adequate and should not be used for identification. Another 50 DOF model is needed, one that actually requires ~ 50 DOF to describe the system fully. Perhaps a 25 story building? Whatever it is, it is important that it pass the Hankel matrix test and that any perturbation should be identifiable when the translational nodes are fully measured.

Why the NTS 5 DOF model doesn't work

The NTS model has the opposite problem – it is just too simple for the structure it is attempting to model. In Appendix 7.17, I show that the SVD decomposition of the Hankel matrix indicated that many more than 5 DOF were required to describe the structure given the recorded outputs. In my final memo (Appendix 7.20), I discuss various suggestions given by the Big Four as to how to make it work, all of which failed. This model is just too simple. There is no easy answer here as to what needs to be done next, but I believe a simpler physical model should be instrumented and recorded so that the problems inherent in identifying physical models may be addressed by a system that is capable of being modeled by 5 or fewer DOF. We need to walk before we run.

Chapter 6. Suggestions for future work

1. Try measuring a simpler physical model with only 3-5 real DOF, like a set of connected springs. This will allow identification of a real system with noisy accelerometer measurements that is much less complex than the NTS structure. I would recommend adding a fourth, smaller mass to the system to explore to what extent model inconsistencies may be accounted for in the designation of Q , the system propagation noise matrix. The nature of the EKF is that some modeling inaccuracies may be compensated for by enlarging Q , but how large the inaccuracies may be and the amount of Q change required is not clear. It would be nice to be able to describe this quantitatively. The NTS structure is too complex to use as the first real structure to be identified.
2. Use optimal smoothing (Gelb, *Applied Optimal Estimation*) to determine the initial conditions of the real state instead of the linear interpolation model now used. This should be more accurate, and it has been reported that the convergence of the algorithm can be heavily dependent on the initial conditions. Or, take data from the modeled system beginning from rest. Of course, this will not be available in the field, so it is perhaps better to do the smoothing.
3. Do not attempt to identify anything using the present 50 DOF model. I have shown that it can be identified with only 10 DOF, and I believe this causes the identification programs to fail. A 50 DOF model is needed that requires 50 DOF to specify the model.
4. It may be interesting to change the 10 DOF model slightly so that it is only 8 DOF, and then see if we can still estimate the system using the 10 DOF model to estimate the 8 DOF system. This would also give us a feel for the amount of system nonlinearities that can be accounted for with Q .

5. It may also be useful to process the measurement data from a real model to get an idea of what order model should be used to simulate the system. There are several ways to do this, using singular value decomposition and others, and it may be useful to look into.
6. One last thing – when setting the noise parameters in `EKF_getnoise.m`, be sure not to use the (parameter)² rule if the parameter is less than 1. Did that once and it wasn't pretty.

6.1 How to get started with the EKF algorithms

The easiest way is just to open up one of my batch test programs like `EKF_test_parsdiff_10DOF.m` in `e:\mfiles\structures\Toolbox\Extended Kalman Filter` and run it. It makes a whole bunch of different perturbations and then simulates the outputs and runs the identifier. You set the measurement noise levels here (for zero noise use – 240 dB to make the algos more stable), as well as the input types (white or sine) and type of the R matrix (used in `EKF_getnoise`). You can also animate the outputs if you want. The results are returned to you in a tabular format and also graphical if you want. It's a good way to get going, as you can see all the different parts of the program.

The other type of batch program is `EKF_test_obsdiff_10DOF.m`. Instead of testing different perturbations, it tests the identification process when different nodes are observed.

Good luck! Call me if you need anything!

Greg

Appendix 7.1. Timeline of development

Dates	Work area
2/5/99	Definition of problem, meeting with Greg, Dave, Matt
2/6 – 2/10	Review of work done to that point (GC, AM files)
2/11 – 2/23	Consolidate damage detection algorithms of GC (use the same variables throughout, comment the files, etc.), write files to read in FEM supplied by Matt
2/24	NTS experiment
2/25 – 2/26	Made all GC script files into function files to save memory, condense files
2/27 - 3/1	Examined continuous to discrete state space transformation, wrote C2DGB.M and C2DCH.M (Cayley-Hamilton theorem, only good for low order systems)
3/2	Discovered RAND.M does not always make white sequences, tested flaw_detection.m for perturbations in K, M, C
3/3 – 3/4	First attempted use of PEM.M (Gauss-Newton iterative gradient search, GN) for identification of simulated 5 DOF system, M and C constrained to be constants
3/5 – 3/8	Examination of Jordan form for F, G, H – rejected as it made G and H full
3/9 – 3/11	Playing with parameters of PEM.M, limited identification success for 5 DOF sim system
3/12	Construction of FIVEDOF_TH.M, which allows me to specify only the independent k_i of K, improving identification accuracy tremendously. Using the changes in k_i only at this point. Identification errors on the order of 1% in only a few iterations.
3/15	Tested changes in K, M, and both K and M for 5 DOF sim, good performance. Wrote flow chart of damage detection and ID algo (structures flowchart.sdr).
3/16 – 3/30	Early work on NTS structure. Constructed a noncausal integrator, attempted to use it to calculate position from accelerator data. Abandoned when noise increases by s^2 .
4/1	First examination of the effects of not having all nodes measured for GN algorithm.
4/2 – 4/14	Construction of TH400DOF.M and associated files, the 400 DOF simulated structure. Detected differences in my model and the ME model, found they are due to roundoff errors in ME model. Started to run PEM for the 4000 DOF system.
4/19 – 6/1	Thorough examination of the GN algorithm, observability, streamlining of PEM.M and associated files into PEM_SPARSE.M. Found out any file that uses sparse matrices

- cannot be compiled, left as sparse. Also determined C2DGB.M was slow and causing errors. Sped up by flushing outer diagonals. Wrote progress report.
- 6/3 – 7/6 Troubleshooting and improving performance of entire GN algo – flaw detection and identification. More Jordan form exploration, Jordan_form.m written.
- 7/12 – 7/13 Received 10 and 50 DOF sim models, built files needed to use them in identification. Came up with and coded the “collapsing” method of handling less than total measurement of all nodes. GN method not built to handle anything less than total measurement of all DOF.
- 7/14 – 7/19 Tests and troubleshooting with 50 and 400 DOF models. Not getting convergence, plus gradient step is much too large. Led to the introduction of limits for both g and $pars$ inside PEM_SPARSE.M.
- 7/21 – 7/23 Examination of the effects of “collapsing” the various identification parameters so that the GN algo will work with reduced measurements. Test of ID algo on sim 50 DOF system. Replaced the determinant criterion (Ljung) with a trace criterion, which is more accurate and stable.
- 7/24 – 8/4 Troubleshooting of ID algo for 5 and 50 DOF. Found that we should be using the calculated value of R_{ee} , not just the default value of $eye(\#DOF)$. Accuracy increased. Also found that my version of \mathbf{K} (the stiffness matrix) and the ME’s version do not match. ME version determined to have roundoff errors, other than that they are the same. Also, since the GN algo is essentially a Wiener filter (the Kalman gain does not change in time, and uniqueness is guaranteed if the system is completely controllable), more research may be needed to determine the best value for the Kalman gain.
- 8/5 – 8/10 Systematic study of the effects of observability on the convergence of the 10 DOF GN ID algo. Locations and number of sensors varied. Results indicate that sensors can do the most good when located at the places where model is poorest. Can actually decrease accuracy if located where model is strong. Repeatability of results also examined, found to be relatively consistent.
- 8/11 – 8/16 Construction of the files needed to identify the NTS 5 DOF model. Ran some preliminary tests, found that fourth floor channel (y12) was missing frequencies above about 40 Hz.
- 8/17 Discovered two problems with ID algo: First, c2dgb heavily dependent on sampling rate. Higher sampling rate, much better conversion. Second, for higher sampling rates, have to

make sure that simulated input has correct frequency components. On some tests where $f_s = 100$ Hz, white noise with components from 0-100 Hz was used, resulting in aliasing and inaccurate results. Fixed both of these problems.

- 8/17 – 8/24 Massive rewrite of entire ID algo so that acceleration could be used as the input to the system. This was quite difficult for the GN algo. Had to redo everything. Also discovered that the white noise input used at NTS did not have enough frequencies below about 6 Hz and extended above 50 Hz to about 52 Hz.
- 8/25 Status report. NTS 5 DOF model will converge given simulated input/output, but only for small perturbations (~5%). Low order sim models (5, 10 DOF) will converge under a wide variety of perturbations and configurations. Higher order sim models (50, 400 DOF) will not converge well at all, 50 DOF will converge somewhat when stiffness values near measured nodes are perturbed.
- 8/26 – 9/6 Examination of the effects of K (Kalman gain) on the stability of the LTTR.M kernel in PEM_SPARSE.M. Tried several different values for K, very few lead to convergence. Also began an in-depth look at the calculation of the initial conditions for the NTS structure. With Larry, came up with a linear interpolation algo that seems to work pretty well for sufficiently high sample rates.
- 9/7 NTS status: Initial conditions works ok, Kalman gain still unsure of, and have to estimate \mathbf{P} from modeled states as we don't have access to actual states (displacement and velocity).
- 9/13 – 9/14 Tutorial "The State of the Art in Vibration-based Structural Damage Identification" in San Jose. Wrote a report on what I learned.
- 9/23 Discovered that NTS 5 DOF model, as given to me, is completely insensitive to changes in stiffness values. Under advice from Dave, tried varying E instead of $E \cdot I$, no good. Finally (10/19) Dave got back word that I should use the shear values α_y as the variables, which worked.
- 9/27 Took a look at Principal Component Analysis (PCA) for use in determining which nodes were the most critical. Results inconsistent, seems to depend on many factors.
- 10/4 Tested sensitivity of whiteness tests on 10 DOF model, could detect damage to pars(3) for changes of 10% or greater.

- 10/5 – 10/9 Tried changing the GN algo so that the direction constant λ (as used in $\theta = \theta_0 + \lambda g^T$) would be a vector instead of a scalar (PEM_SPARSE.VECTORK.M). It was hoped that this would make the GN algo more sensitive to changes in a single stiffness value in a multiple DOF system. Results ambiguous – algorithm now takes about twice the time to run, only slightly more accurate. Decided it wasn't worth the effort.
- 10/11 Gave report on the project to the LDRD review board.
- 10/15 Reviewed the data from the NTS experiment. Compared the psds of the output files in the y direction (y3, y6, y9, y12, y15) for all of the white noise, undamaged tests (strucow#.mat). Results all consistent – same resonance locations, all have frequencies above about 35 Hz missing from y12.
- 10/19 - 10/20 Changed NTS ID algorithm to operate on α_y instead of EI. Identification results mixed – Sort of converged to the desired efreq of 4.5, 14.7, and 31.9 Hz. Parameters changed by a reasonable amount – from 1% to 14%.
- 10/23 - 11/7 Vacation, JASA conference in Ohio
- 11/8 – 11/26 I have decided to change the ID algorithm from the GN approach to an optimal filter approach. This should have better noise robustness, convergence, and not all nodes will have to be measured. We will also be able to use acceleration directly as a measurement as well as operate the state propagation in the continuous domain, obviating the need for the cont – discrete transformation that has caused so many problems. Spent this time familiarizing myself with all types of Kalman filters and estimators.
- 11/30-12/22 Construction and troubleshooting of 1-D continuous/discrete Extended Kalman Filter (EKF) algorithm (EKFSHO.M). Works very well, converges in about 20-30 samples.
- 1/3 – 1/5/00 Examination of the effects of the various noise parameters on the convergence of the EKF. Wrote report.
- 1/6 – 1/12 Extension of the 1-D EKF to multiple DOF. This involves the calculation of both F and H symbolically. This can take quite a long time, but for each model only has to be done once and then can be recalled and the present state values inserted. Took some doing, using strings and symbolics together to speed up the process.
- 1/13 – 1/23 Condensing and speeding up EKF_ID.M. Went from 4.7 hours/1000 samples to 3.2 hours. Tried to speed up by not calculating P if ΔP was below a threshold for 10-20

samples, didn't work as DP can be quite small for a few samples then jump back up to substantial values.

- 1/24 Compared the performance of the EKF to the GN for a variety of sensor locations and number of sensors. Wrote report.
- 1/25 Tried simple approximation for Pdot calculation in EKF, hoping to speed things up. Didn't work. Also began processing NTS data with EKF, no success. No convergence or the α_y values driven to lowest value allowed.
- 1/26 Checking the NTS noise parameters for correct size, tried a few different ones. Still no convergence.
- 1/31 – 2/3 Many different methods tried to speed up Pdot calculation. Profiled several different methods, compiled different files, final results pretty good. Ended up compiling VEC2MATS.M and MAT2VECS.M, then gutting ODE45.M and getting rid of everything that wasn't needed (ODE45_GB.M). Processing time for 10 DOF dropped from 3.2 hours for 1000 samples to 42 minutes. Now it is actually usable.
- 2/4 Coded the files necessary for the 50 DOF model to be identified by EKF. Coded the files needed so that all of the attributes (α_y , E, and L) of the NTS model could be used as parameters.
- 2/7 Fundamentally changed the EKF algo so that all the members of the stiffness matrix **K** (not just the independent values, but not including every member as **K** is symmetric) could be used as identification parameters. Saved files with suffix _all_k.
- 2/7-2/24 Ran several different configurations (with and without noise) for 10 DOF, 50 DOF, and NTS 5 DOF using both the normal and the _all_k files. The _all_k 10 DOF parameters that represented combinations of independent parameters converged, but the others did not. The NTS 5 DOF _all_k model did not converge at all, even after analyzing 50 seconds of data. The 10 DOF model did rather well in the presence of -10 and -20 dB noise, especially with full measurements. 50 DOF did not converge even without noise. Tried increasing the beginning value of **P**, hitting it harder so that the displacements would be larger (helped a little), among others. Wrote "EKF results for 10, 50, and NTS 5 DOF".
- 2/25-2/29 Examined the SVD decomposition of the generalized Hankel matrix as used for determining correct model order. Found that the 10 DOF system required only 6-8 DOF

to be described adequately. Similarly, the 50 DOF system needed only 13 DOF (and perhaps as few as 10 DOF) to be described. It is therefore vastly over-specified.

Conversely, the NTS 5 DOF system was shown to be vastly underspecified, needing more than 25 DOF to adequately describe the motion. Wrote “Estimating model order using the SVD decomposition of the generalized Hankel matrix”.

- 3/1/00 Had advisory meeting with Dave McCallen, Dave Harris, and Jim Candy to discuss problems above. They offered several suggestions, outlined in memo “Suggestions and comments from advisory meeting of 3/1/00”.
- 3/2 – 3/14 Implementing several suggestions including the ones concerning the Pdot calculation. These included skipping a calculation (one member of \mathbf{P}) if ΔP_{ii} dropped below a threshold. It wasn’t kept constant, though, it was re-examined after the next update. This worked relatively well, although the threshold had to stay at 1% or less to ensure convergence. Even then, convergence took longer than with the normal Pdot calculation. Also wrote file that allowed examination of ΔP , ΔX , ΔK , and ΔY as a function of time. Discovered that only ΔX (stiffness states only) and ΔY go to zero as a function of time, and the others do not. Use the DX metric to stop calculations – if DX is less than 0.2% 20 times in a row, the program terminates.
- 3/15 Successfully identified the 50 DOF simulated output using the 10 DOF model. Shows that 50 DOF model is overspecified, needs to be redone.
- 3/16-3/17 Wrote up results from above experiments
- 3/20-3/21 Began the process of using a series of notch filters to remove information from the recorded NTS data so that the NTS 5 DOF model may be able to identify it properly. Built first attempt at a filter, examined the psds of the recorded data as well as the innovations of the ID process when no filter was used.
- 3/22-3/24 One more attempt to speed up \mathbf{P} calculation process. I thought it might be possible to only calculate the Pdot for the stiffness values, as that is what we are interested in identifying. They only make up a small fraction of the total size of \mathbf{P} , so that would speed up the Pdot calculation considerably. This didn’t work that well (mean errors on the order of 30%), so I threw in calculating all of Pdot every 5th or 10th sample. This would still speed things up, without sacrificing too much accuracy. This worked much better, with mean accuracies down to 1.1% and 3.4%, respectively. Surprisingly, it works even better if the stiffness

values are not calculated in between calculating all of Pdot. Calculating Pdot by itself every 10 samples resulted in a mean error of only 1-2%, with a reduction of 90% in Pdot calculations. This was the strategy adopted for systems with 10 or more DOF.

3/27 – 4/11 Back to the NTS data. After Chad has run my recorded input through his simulator and we confirmed that the psds were the same, I tried to identify it using the simple filter of 3/20 (EKF_BP.M). The results were still quite poor. I then built a series of Chebychev II notch filters (EKF_notch.m), in order to more thoroughly remove the frequency components of disinterest. This too, failed to produce reasonable values for the α_y values, although at times the resonance locations seemed to be close to those desired. This was probably just a coincidence.

4/12 – 5/1 Writing the final report, consolidating records, preparing computer files for transfer to next PI.

Appendix 7.2 File Locations

The files of interest are all on the NT machine, the Dell P2-350. It is not that fast but has plenty of RAM and disk space for backups. The main Matlab files are all in e:\Mfiles, with several sub-folders. The main one for this project is \structures, which contains several more folders. The main ones there are \GB Files, which contains a lot of my miscellaneous files, \Toolbox, which contains most of the GN identification algorithms, \Toolbox\Extended Kalman which contains the EKF files, \Models which contains all the model information, and \AM Mfiles which has some files useful for ME to EE translation. Also of interest is \GB Files\EKF_results, which has a lot of the results discussed in this paper in .txt and .doc form.

The documents of interest can be found in e:\Documents, with this paper in \Final structures\report and the various memos in \Progress Reports\Structures. There are some of my Powerpoint presentations in \Ppt Presentations, and some presentation graphics can be found in e:\Presentation Graphics.

These files are all backed up daily onto the F: drive using FileBack PC, a shareware program.

The NTS structure data files are located in G:\Data\Structures, but you shouldn't need to access them as the corresponding .mat files are on the NT machine in e:\mfiles\structures\matfiles.

Appendix 7.3 References

1. Lennart Ljung (1987). "System Identification theory for the user". Prentice-Hall, Englewood Cliffs, NJ. ISBN 0-13-881640-9
2. J.E. Dennis and R.B. Schnabel (1983), chapter 10. "Numerical Methods for Unconstrained Optimization and Nonlinear Equations", Prentice-Hall, Englewood Cliffs, NJ. ISBN 0-13-627216-9
3. Gelb, A. (ed. 1999). "Applied Optimal Estimation", The M.I.T. Press, Cambridge, MA. ISBN 0-262-57048-3.
4. Franklin, G.F., Powell, J.D., Workman, M.L. (1990). "Digital control of dynamic systems". Addison-Wesley Publishing Company, Reading, MA. ISBN 0-201-11938-2
5. Kallstrom, C. (1973). "Computing $\text{EXP}(A)$ and the integral of $\text{EXP}(As) ds$ ". Report 7309, Lund Institute of Technology, Division of Automatic Control, March 1973. *Not yet located.*

Appendix 7.4. Filenames and descriptions

Name	Function
!t.m	Template file for new files
animate5.m	Animates 5 outputs for a five story building
c2dgb.m	GB file to transform the continuous matrices F and G to discrete form (GN)
c2dgb400.m	GB file to transform the continuous matrices F and G to discrete form for 400 DOF (GN)
c2dgb5.m	GB file to transform the continuous matrices F and G to discrete form for 5 DOF (GN)
collapse.m	Collapses a matrix by removing rows specified in obsmat (GN)
compare_NTS_psd.m	Compares the psds of the output of the NTS structure
Discrete_Model_Builder.m	Function for building the continuous and discrete models for the structures project. (GN)
EKF_10MCK.m	To specify a 10 DOF state-space model for identification purposes. The free variables (pars) are the E values for each element.
EKF_10MCK_all_k.m	SAA except free variables now the K_{ij}
EKF_1MCK.m	SAA for 1 DOF (SHO)
EKF_50MCK.m	SAA for 50 DOF
EKF_5MCK.m	SAA for 5 DOF (NTS).
EKF_5MCK_all_k.m	SAA for 5 DOF (NTS) and free variables now the K_{ij} .
EKF_Algo.m	Main EKF algorithm.
EKF_BP.m	Returns filter coefficients for simple bandpass filter (use EKF_notch instead)
EKF_calcFH.m	Calculates the F and H matrices for the extended Kalman filter for a system of any DOF
EKF_calcPJ.m	Calculates the Jacobian for N DOF systems and determines if it is constant, then saves output info (J F indF cflag) in EKF_J(Mord) (not used anymore!)
EKF_getnoise.m	Returns the noise matrices (Pminus, Q, Rk) for the different models for the EKF
EKF_ID.m	Main calling program for the simulated system EKF identification
EKF_NTS.m	Main calling program for the real NTS system EKF identification
EKF_NTS_Music.m	Recalls NTS data and plots MUSIC information as well as psds of the recorded input and output and simulated output
EKF_Pprop.m	Supplies the information necessary to propagate P in EKF_Algo

EKF_Pprop.compile.m	Compilable version of the above. Did not increase speed, not used.
EKF_Pprop_part.m	Same as above but only propagates the stiffness part of the equation. Did not help, not used
EKF_sim.m	EKF simulation engine. Used to simulate outputs of a model with real or simulated input.
EKF_test_obsdiff_10DOF.m	Tests the 10 DOF model with the number and location of measurements varied, runs on NT machine
EKF_test_obsdiff_10DOF_2.m	SAA, but used on 98 machine
EKF_test_parsdiff_10DOF.m	Tests the 10 DOF model with the number and location of stiffness perturbations varied, no change in measured nodes, runs on NT machine
EKF_test_parsdiff_10DOF_all_k.m	SAA, uses K_{ij} as parameters
EKF_test_parsdiff_10DOF_noise.m	Tests the 10 DOF model by changing the amount of measurement noise, pars and measurement locations constant, 98 machine
EKF_test_parsdiff_50DOF.m	Tests the 10 DOF model with the number and location of stiffness perturbations varied, no change in measured nodes, runs on NT machine
EKF_test_parsdiff_50DOF_2.m	SAA, runs on 98 machine
EKF_Xprop.m	Supplies the information necessary to propagate P in EKF_Algo
EKF_Xprop_compile.m	Compilable version of the above. Did not increase speed, not used
EKFSHO.m	EKF using a simple harmonic oscillator (1 DOF). Good to get a feel for the EKF algorithm.
errint.m	Calculates the error vs. frequency for a digital integrator (GN)
ev400.m	Calculates the eigenvalues for the 400 DOF system (GN)
fileview for NTS.m	Views the scaled NTS data
findflaw.m	Damage identification algorithm for GN, called after flaw_detection.m
fiveDOF.m	Builds state-space model for the 5 DOF (not NTS) model (early)
fiveDOF_sparse.m	SAA using sparse matrices
fivedof_th.m	Builds theta model for the 5 DOF (not NTS) model (calls five_dof.m)
flaw_detection.m	This is a supervisory code to (1) Simulate the structure dynamics and (2) Run a Kalman Filter to estimate states, and (3) detect flaws by performing a whiteness test on the innovations
flush.m	Removes matrix elements below a threshold. Used to remove rounding errors (by W. Tych).
gbf.m	Changes to the L:\Mfiles\Structures\GB Files directory from the 98 machine
gbf.m	Changes to the E:\Mfiles\Structures\GB Files directory from the NT machine
getaux.m	Gets the auxiliary variables for 400 DOF

<code>getNTSRv.m</code>	Calculates R_v (measurement noise covariance matrix) using recorded NTS data background files
<code>getpars1.m</code>	Gets the default estimate parameters for the SHO 1 DOF system
<code>getpars10.m</code>	Gets the default estimate parameters for the 10 DOF system
<code>getpars10_all_k.m</code>	Gets the default estimate parameters for the 10 DOF system when the K_{ij} are used as the parameters
<code>getpars400.m</code>	Gets the default estimate parameters for the 400 DOF system
<code>getpars5.m</code>	Gets the default estimate parameters for the 5 DOF (not NTS) system
<code>getpars50.m</code>	Gets the default estimate parameters for the 50 DOF system
<code>getparsNTS5.m</code>	Gets the default estimate parameters for the NTS 5 DOF system
<code>getparsNTS5_all_k.m</code>	Gets the default estimate parameters for the NTS 5 DOF system when the K_{ij} are used as the parameters
<code>Hankel_test.m</code>	For the simulated models (10 and 50 DOF), this computes the observability matrix O and controllability matrix W used to calculate the general Hankel matrix H . This SVD of this is calculated and the values checked against a threshold to determine the minimum order of the system.
<code>Hankel_test_NTS.m</code>	SAA for the NTS model
<code>issymm.m</code>	Determines if the input matrix is symmetric
<code>Jordan_form.m</code>	This routine reads in the matrices from a discrete-time LTI system state space model F_d, G_d, H_d and performs a similarity transformation on them to create a "normal form" system. (GN, makes F diagonal but G and H full, not used)
<code>kb.m</code>	The short version of keyboard, saves time
<code>kbeam.m</code>	Returns the local k matrix for a standard beam element
<code>kcolumn.m</code>	Returns the local k matrix for a standard column element
<code>kcolumn10.m</code>	Returns the local k matrix for a column element for the 10 DOF model
<code>kcolumn50.m</code>	Returns the local k matrix for a column element for the 50 DOF model
<code>kcolumnNTS5.m</code>	Returns the local k matrix for a column element for the NTS 5 DOF model
<code>Kf.m</code>	This is the Kalman Filter routine used to perform the whiteness test in <code>flaw_detection</code> (GN, G. Clark)
<code>kf_caller.m</code>	This calls the above (GN, G. Clark)
<code>ltitr_comp.m</code>	Compiled version of Matlab's <code>ltitr.m</code> , a linear time-invariant time response kernel. (GN, not used)
<code>ltitr_sparse.m</code>	Version of Matlab's <code>ltitr.m</code> that can handle sparse matrices. (GN)
<code>mat2str.m</code>	Extension of <code>num2str.m</code> (W. Tych)
<code>mat2vecc.m</code>	Vectorizes matrices by columns

<code>mat2vecs.m</code>	Vectorizes symmetric matrices by columns (EKF)
<code>matdiff.m</code>	Determines where and how severe the difference between two matrices is located. Designed to test K construction.
<code>mf2th_sparse.m</code>	Sparse version of Matlab's <code>mf2th.m</code> . (GN)
<code>model_est.m</code>	This code sets up the "true" Gauss-Markov system model to be simulated and passes it to the state estimator. (GN, G. Clark)
<code>NTS2ascii.m</code>	File to read in NTS data, convert to g's, and save as .mat files for me and ASCII files for MEs
<code>NTSff.m</code>	Loads the NTS files of choice, runs <code>findflaw</code> to do damage detection (GN)
<code>NTS_init_cond.m</code>	Calculates the initial conditions of the NTS structure using interpolation (EKF)
<code>NTS_plot_out.m</code>	Plots the psds of the outputs from the ME's NTS model
<code>NTS_sens.m</code>	Calculates the sensitivity of the NTS mode frequencies to changes in E
<code>nuderstgb.m</code>	GB's version of <code>nuderest.m</code> , which selects the step size for numerical differentiation (GN)
<code>Observability_test.m</code>	Computes the observability matrix Q used to test for "complete observability" of a discrete-time linear time-invariant system. (GN, G. Clark)
<code>ode45_gb.m</code>	Seriously gutted version of <code>ode45</code> . It was gutted to improve the memory requirements and speed of calculating <code>Pprop</code> for <code>EKF_algo</code> . Wildly successful. (EKF)
<code>onedtest.m</code>	1-D EKF testbed, early version of <code>EKF_SHO.m</code>
<code>Pchange.m</code>	3-D representation of the change in P per sample for the EKF
<code>pem_sparse.m</code>	Rewrite of Matlab's <code>pem.m</code> routine so that it is more efficient and able to work with sparse matrices. Only works with theta form. (GN)
<code>pem_sparse_constantk.m</code>	SAA but used different version of Kalman gain (GN, not used)
<code>pem_sparse_symbolic.m</code>	Attempted (unsuccessfully) to run the above symbolically (GN)
<code>pe_sparse.m</code>	Sparse version of Matlab's <code>pe.m</code>
<code>r2s.m</code>	Changes real matrices into symbolic ones with all nonzero values converted to symbolic terms
<code>readelem.m</code>	Reads in ASCII files with model elemental information and save it in .mat form. Very useful when adding new models!
<code>readnodes.m</code>	Reads in ASCII files with model node information and save it in .mat form. Also very useful when adding new models!
<code>read_accel.m</code>	Reads in ASCII files with acceleration info in 2nd column and save it in .mat form. MEs usually supply time in first column and acceleration in second.

<code>ret_efreq.m</code>	Calculates the eigenfrequencies given M, K – centralizes the calculation needed to build C
<code>SHOacc.m</code>	Converts displacement and velocities into acceleration given a damped SHO system
<code>shop.m</code>	SHO P propagation
<code>shox.m</code>	SHO X propagation
<code>SIM_Discrete.m</code>	Supervisory routine that simulates the output of a given model. (GN, G. Clark)
<code>sparseit.m</code>	Examines a matrix as sparses it if it is sparse enough
<code>sqrtslow.m</code>	Perturbed version of Matlab's <code>sqrtn.m</code> (GN)
<code>subs_sym.m</code>	Speeds up the subs routine in the symbolic toolbox by using the symmetric properties of the matrices to be substituted
<code>th10DOF.m</code>	Specifies a 10 DOF state-space model using the parameters given (GN).
<code>th10DOFMCK.m</code>	SAA, only returns the M, C, and K matrices (EKF)
<code>th400DOF.m</code>	Specifies a 400 DOF state-space model using the parameters given (GN).
<code>th400DOF_big_K.m</code>	SAA, but uses the K_{ij} as the parameters. (GN)
<code>th400dof_caller.m</code>	Constructs the theta model using <code>th400DOF.m</code> (GN)
<code>th400DOF_small_k.m</code>	Original version of <code>th400DOF.m</code> (GN, not used)
<code>th50DOF.m</code>	Specifies a 50 DOF state-space model using the parameters given (GN)
<code>thNTS5DOF.m</code>	Specifies a NTS 5 DOF state-space model using the parameters given (GN)
<code>thNTS5DOFMCK.m</code>	SAA, only returns the M, C, and K matrices (EKF)
<code>thNTS5DOF_init_cond.m</code>	SAA, only the free variables are the initial conditions
<code>vdpgems.m</code>	Runs the <code>vdcode</code> so that it resembles the GEMS output
<code>vdcode_gb.m</code>	My own specification of the VDP ODE
<code>vec2matc.m</code>	Turns vectorized column matrices back into matrices
<code>vec2mats.m</code>	Turns vectorized symmetric column matrices back into matrices
<code>Whiteness_Test.m</code>	Performs statistical whiteness tests on a given signal (G. Clark)
<code>wng.m</code>	White noise generator, use <code>idinput.m</code> instead.
<code>Wssr.m</code>	Performs the Weighted Sum Squared Residual (WSSR) test on an innovations vector (G. Clark)

AM Files – these were originally written by A. Meyer, modified in some way by me.

<code>Files2accel.m</code>	Reads in acceleration data from ME ASCII data files and translates into matrices. Calls <code>Files2Matsaccel.m</code> .
<code>Files2accel25.m</code>	SAA but with 25 files.

Files2Matrices.m	Reads in model data from ME ASCII data files and translates into matrices. Called by Files2MCK.M.
Files2Matsaccel.m	Child of Files2accel.m
Files2Matsaccel25.m	Child of Files2accel25.m
Files2MCK.m	Major file used to convert the ME model descriptions into something we can use. Calls Files2Matrices.m.

Backup copies are made at regular intervals onto the C drive of the NT machine. Old versions are saved in OLD folders with the version number appended after the original ending. For example, version 2.2 of EKF_ID.m is saved as EKF_ID.22.m.

Interdepartmental letterhead
 Mail Station: L-271
 Ext: 33088

3/9/99

To: Greg Clark
 Jim Candy
 Dave McCallen
 Matt Hoehler

From: Greg Burnett

CC: Larry Ng

Re: System ID algorithm

Hello everyone,

I thought I would put a memo together to outline the algorithm I am using for system identification. The problem is this: we want to identify discrepancies between a "best guess" model and the actual model. The differences could be due to modeling inaccuracies, or by damage to an already-modeled structure. This procedure starts with the "best guess" model of the system and proceeds to calculate an improved model that corresponds best (in an innovation least-squares sense) to a given input and output. Changes in the model or the structure can be determined by observing the differences between the "best guess" model and the improved model.

The algorithm I have constructed so far is this:

- 1) We are given a "best guess" model of the system by the ME guys. In this memo I will use the 5 DOF system. This "best guess" model consists of K and M matrices, which we use to calculate a C matrix. The M, C, K matrices are then used to form the continuous F, G, and H matrices of the state-space model. The matrices are as follows:

$M = 1.0e+003 *$

5.0450	0	0	0	0
0	5.0450	0	0	0
0	0	5.0450	0	0
0	0	0	5.0450	0
0	0	0	0	3.8099

University of California



**Lawrence Livermore
 National Laboratory**

K = 1.0e+006 *

3.2443	-1.6222	0	0	0
-1.6222	3.2443	-1.6222	0	0
0	-1.6222	3.2443	-1.6222	0
0	0	-1.6222	3.2443	-1.6222
0	0	0	-1.6222	1.6222

C = function of M and K

F =

0	0	0	0	0	1.00	0	0	0	0
0	0	0	0	0	0	1.00	0	0	0
0	0	0	0	0	0	0	1.00	0	0
0	0	0	0	0	0	0	0	1.00	0
0	0	0	0	0	0	0	0	0	1.00
-643.08	321.54	0	0	0	-223.77	89.15	0	0	0
321.54	-643.08	321.54	0	0	89.15	-223.77	89.15	0	0
0	321.54	-643.08	321.54	0	0	89.15	-223.77	89.15	0
0	0	321.54	-643.08	321.54	0	0	89.15	-223.77	89.15
0	0	0	425.78	-425.78	0	0	0	118.05	-163.53

G = 1.0e-003 *

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0.1982	0	0	0	0
0	0.1982	0	0	0
0	0	0.1982	0	0
0	0	0	0.1982	0
0	0	0	0	0.2625

H =

1	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0

- 2) The F matrix (which depends on M, C, and K) and the G matrix (which depends only on M) are modified into A, B, and C to conform with Matlab's notation by inserting NaN wherever a parameter can be thought of as variable. For this first test, only K was modified, so only the nonzero values in F are changed to NaN.

A =

0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	1
NaN	NaN	0	0	0	NaN	NaN	0	0	0
NaN	NaN	NaN	0	0	NaN	NaN	NaN	0	0
0	NaN	NaN	NaN	0	0	NaN	NaN	NaN	0
0	0	NaN	NaN	NaN	0	0	NaN	NaN	NaN
0	0	0	NaN	NaN	0	0	0	NaN	NaN

B = G;

C = H;

- 3) The nonzero values that were replaced in A are saved in a vector "init_param" to use as the initial parameters for the identification.

```
init_param = -643.08, 321.54, -223.77, 89.15, 321.54, -643.08, 321.54, 89.15, -223.77, 89.15,
321.54, -643.08, 321.54, 89.15, -223.77, 89.15, 321.54, -643.08, 321.54, 89.15, -223.77, 425.78, -
425.78, 118.05, -163.53
```

If you compare `init_param` to `F`, you will see that the values are read row by row to be used as initial values.

4) An input (`u`) and output (`y`) matrix are formed into the `z` matrix using:

```
z = [r2c(y(1,:)) r2c(y(2,:)) r2c(y(3,:)) r2c(y(4,:)) r2c(y(5,:))
      r2c(u(1,:)) r2c(u(2,:)) r2c(u(3,:)) r2c(u(4,:)) r2c(u(5,:))];
```

where `r2c` changes the row to a column vector. The output can be from either actual measurements or through simulations (such as using `lsim.m` from the control toolbox in `Discrete_Model_Builder`). The simulations are done using the Gauss-Markov state propagation outlined in Jim's book, p. 33. The three noise covariance matrices are the ones specified by Greg in his report, with R_v and R_w recalculated after the random noise is generated. The input is white noise, on the fifth floor, with a T_s of 0.01 and a length of 25 seconds.

5) The initial values for the states (`x0`) is set to all zeros.

```
x0 = zeros(Nx,1); % initial state vector
```

6) The noise matrix `W` is formed as

```
W = eye(ra,rc); % noise autocorrelation
```

where `ra` and `rc` are the number of rows in `A` and `C`, respectively.

7) The model structure is formed using the following commands:

```
ms = modstruc(A, B, C, D, W, x0);
th = ms2th(ms, 'czoh', init_param, Ree(:, :, end), Ts);
% THETA = MS2TH(MS, CD, PARVAL, LAMBDA, T)
```

where `lambda` is the covariance matrix of $e(t)$, R_{ee} , and we use the last value. This is calculated by `flaw_detection.m`.



8) The parameter estimation is performed by:

```
[thnew, iterinfo] = pem(z, th, [], 100, [], [], [], Ts, 'trace');
% [TH, IT_INF] = PEM(Z, THSTRUC, index, maxiter, tol, lim, maxsize, T)
```

9) The new estimate for theta is returned to state-space so that the differences can be compared:

```
[Fn Gn Hn temp Wn x0] = th2ss(thnew); % back to state space from theta
```

As an example, I have taken the original K matrix and perturbed it so that k_3 has been reduced to 50% of its previous value. This changes K so that it is now

K = 1.0e+006 *

3.2443	-1.6222	0	0	0
-1.6222	2.4332	-0.8111	0	0
0	-0.8111	2.4332	-1.6222	0
0	0	-1.6222	3.2443	-1.6222
0	0	0	-1.6222	1.6222

I then ran `flaw_detection.m` with a damping coefficient of 5% to see if a flaw was discovered. `Whiteness_test.m` returned the following results for each innovation:

E(1): ***** White *****

Percent out of bounds = 4.32.

E(2): ##### Non-White #####

Percent out of bounds = 23.10.

E(3): ##### Non-White #####

Percent out of bounds = 35.17.

E(4): ##### Non-White #####

Percent out of bounds = 6.08.

E(5): ##### Non-White #####

Percent out of bounds = 5.52.

This indicates that something is wrong with the model around states 2 and 3, especially state 3. This corresponds well to the known deficiency. Now we pass R_{ee} , u , and y to `findflaw.m` and see if it can detect the change in K . After 16 iterations, it derived the following, where `pmodelK` is the perturbed model of K , `calcK` is `pem.m`'s calculated estimate of K , and `diffK` is the difference between the two:

`pmodelK = 1.0e+006 *`

3.2443	-1.6222	0	0	0
-1.6222	2.4332	-0.8111	0	0
0	-0.8111	2.4332	-1.6222	0
0	0	-1.6222	3.2443	-1.6222
0	0	0	-1.6222	1.6222

`calcK = 1.0e+006 *`

3.1361	-1.5372	0	0	0
-1.6487	2.3961	-0.7204	0	0
0	-0.9362	2.5078	-1.7339	0
0	0	-1.4930	3.1496	-1.5480
0	0	0	-1.7302	1.6792

`diffK = modelK - calcK = 1.0e+005 *`

1.0821	-0.8492	0	0	0
0.2656	0.3719	-0.9071	0	0
0	1.2512	-0.7456	1.1179	0
0	0	-1.2918	0.9473	-0.7418
0	0	0	1.0799	-0.5703



iterinfo =16.000 -0.0011 0.0068 (last iteration, last fit improvement, norm of last search vector)

This is quite encouraging, as the changes to K shown in pmodelK are mirrored quite well in the calculated calcK. I will continue to test this algorithm, and when I am confident of its abilities I will use it to determine the changes Matt made in M and K for the 5 examples he provided me.

Still to do:

- 1) Allow M to vary as well, see if the solution still converges.
- 2) Restrict the variation of the parameters to those associated with the non-white states. For example, in this case states 2 and 3 were the only ones to exhibit significant non-whiteness. Therefore perhaps we would only let pem.m treat as variables the parts of F that correspond to states 2 and 3. This may facilitate convergence of the solution.



Interdepartmental letterhead

Mail Station: L-271

Ext: 33088

3/15/99

To: Greg Clark
Jim Candy
Dave McCallen
Matt Hoehler

From: Greg Burnett

CC: Larry Ng

Re: Flaw detection and identification algorithm (5 DOF)


Hello everyone,

After my last memo, I got several excellent suggestions from Jim and Dave and was able to construct a system identifier that works quite well. I have incorporated it into the previous flaw detector, and now have a complete flaw detection and identification algorithm that works very well for the five degree of freedom problem. This memo is to explain the algorithm and to once again solicit comments.

The layout of the algorithm is shown in Figure 1, presented at the end of this memo. It will be helpful to follow along with it as we go.

- 1) In FLAW_DETECTION, we enter the model order and initialize the time vector to be used to do the testing and simulation. We can simulate the output of a system (using normal or perturbed M and K matrices) using SIM_DISCRETE, which uses MF2TH (described below) to construct F_d , G_d , and H_d (the discrete versions of F, G, and H, our system matrices). An appropriate input is loaded (in this case (white noise * 1000) at the fifth floor with a T_s of 0.01 and a length of 25 seconds). It then uses Jim's Gauss-Newton progression algorithm with system and measurement noise to simulate a noisy output (y_{sim}). One difference I inserted here was to recalculate R_v and R_w after they had been created. It gives a more accurate picture of the covariance functions of the noise. If an actual

University of California

 **Lawrence Livermore
National Laboratory**

measurement or test series is available, the corresponding u and y matrices are loaded and the simulation section is not used.

- 2) The simulated or actual inputs and outputs are fed into KF_CALLER. It first calls MODEL_EST, which loads the original (unperturbed) M and K matrices and passes them to DISCRETE_MODEL_BUILDER, which returns the matrices to be used for Kalman estimation: F_{de} , G_{de} , and H_{de} . Note that we can use the same method (MF2TH), but I used the older way to make sure the two were compatible.
- 3) The matrices R_w , R_v , P_o , x_{init} , and F_{de} , G_{de} , and H_{de} are all passed to KF, which uses the Kalman filter to predict the outputs (y_p). The innovations (errors in predicted output, $y - y_p$) are recorded and sent to a whiteness tester, in this case WHITENESS_TEST but for larger models it will be WSSR. If the innovations are considered non-white, FINDFLAW is launched to determine where the faults in the model are.
- 4) In FINDFLAW, the original M and K are loaded. This is the “best guess” model of the system determined by Dave and Matt. We use the K and M matrices and a damping constant to calculate a C matrix. The M , C , K matrices are then used to form the continuous F , G , and H matrices of the state-space model. The matrices are as follows:

$M = 1.0e+003 *$

5.0450	0	0	0	0
0	5.0450	0	0	0
0	0	5.0450	0	0
0	0	0	5.0450	0
0	0	0	0	3.8099

$K = 1.0e+006 *$

3.2443	-1.6222	0	0	0
-1.6222	3.2443	-1.6222	0	0
0	-1.6222	3.2443	-1.6222	0
0	0	-1.6222	3.2443	-1.6222
0	0	0	-1.6222	1.6222

$C =$ function of M and K



I constructed a function called FIVEDOF.M which has the following form:

```
[A,B,C,D,K,x0] = fiveDOF(pars,T,aux);
```

The input arguments are as follows: Pars are the parameters to be estimated, in our case they are the changes to the values (dm and dk) of the underlying m1, m2, m3, m4, m5 and k1, k2, k3, k4, k5 values that construct the matrices M and K. T is the sampling time for the matrices, it is zero for continuous matrices. Aux contains auxiliary terms, in this case m1...m5 and k1...k5. FIVEDOF takes these values and constructs the A, B, C, D, K, and x0 matrices necessary for our application (this is not K the stiffness matrix, it is K the noise matrix, using Matlab's notation). If discrete matrices are needed, it discretizes them using C2D. In this way FIVEDOF limits the number of unknowns to 10 (5 m and 5 k values) and forces the correct symmetric form for F and G. Right now an x0 of all zeros and a K of

K =

```
1  0  0  0  0
0  1  0  0  0
0  0  1  0  0
0  0  0  1  0
0  0  0  0  1
0  0  0  0  0
0  0  0  0  0
0  0  0  0  0
0  0  0  0  0
0  0  0  0  0
0  0  0  0  0
```

are used.

5) An input (u) and output (y) matrix are formed into the z matrix using:

```
z = [r2c(y(1,:)) r2c(y(2,:)) r2c(y(3,:)) r2c(y(4,:)) r2c(y(5,:))
     r2c(u(1,:)) r2c(u(2,:)) r2c(u(3,:)) r2c(u(4,:)) r2c(u(5,:))];
```

where r2c changes the row to a column vector.

6) The model structure is formed using the following command:

```
thtrue = fivedof_th(M,K,Ree(:, :, end), dk, dm);
% returns "true" theta modal used for the experiment
```

where k_0 and m_0 are calculated and given to mf2th

```
m = diag(M); % M already diag
k0 = K(5,5); % nominal value for the k's

% now for th - uses form TH = MF2TH(MODEL,CD,PARVAL,AUX,LAMBDA,T)
% parval are the things we are changing, aux is nominal values, aux(11) = percent
damping

th = mf2th('fivedof','c',[dk dm],[k0 k0 k0 k0 k0 c2r(m) 5], Ree, 0);
```

and where $dm = dk = [0 \ 0 \ 0 \ 0 \ 0]$ for the unperturbed system. The last value of aux is the damping coefficient in percent. Lambda is the covariance matrix of $e(t)$, R_{ee} , and we use the last value. This is calculated by KF.

7) The parameter estimation is performed by:

```
[thnew, iterinfo] = pem(z,thtrue,[1:5],100,[1e-12],[[]],[[]],Ts,'trace');
% [TH,IT_INF] = PEM(Z,THSTRUC,index,maxiter,tol,lim,maxsize,T)
% index = [1:5] = only estimate first five unknowns
```

where we have further limited the pars to only the first five values, the dk vector. Thus we are assuming only the values of k change for this example. I will address changing the m values and changing both below.

8) The new estimate for theta is returned to state-space so that the differences can be compared:

```
[Fn Gn Hn temp Wn x0] = th2ss(thnew); % back to state space from theta
```

The lower half of G_n yields M, and the lower right fourth of F_n yields K.

As an example, I have taken the original K matrix and perturbed it so that k_1 has been reduced by 87.5%, k_2 has been reduced by 75%, k_3 has been reduced by 10%, k_4 has been reduced by 33%, and k_5 has been reduced by 66% of its previous value. This changes K so that it is now

$K = 1.0e+006 *$

0.6083	-0.4055	0	0	0
-0.4055	0.5678	-0.1622	0	0

University of California



**Lawrence Livermore
National Laboratory**

0	-0.1622	1.2437	-1.0814	0
0	0	-1.0814	1.6222	-0.5407
0	0	0	-0.5407	0.5407

I then ran `flaw_detection.m` with a damping coefficient of 5% to see if a flaw was discovered. `Whiteness_test.m` returned the following results for each innovation:

E(1): ##### Non-White #####

Percent out of bounds = 90.7274.

E(2): ##### Non-White #####

Percent out of bounds = 91.0472.

E(3): ##### Non-White #####

Percent out of bounds = 80.7354.

E(4): ##### Non-White #####

Percent out of bounds = 87.6099.

E(5): ##### Non-White #####

Percent out of bounds = 78.1775.

In this case, unlike the earlier one where the damage was localized, the whiteness test is not terribly useful, as there is much damage over the entire structure.

Now we pass R_{ee} , u , and y to `FINDFLAW` to see if it can detect the change in K . After 9 iterations, it derived the following, where `pmodelK` is the perturbed model of K , `calcK` is PEM's calculated estimate of K , `diffK` is the difference between the two, and `percK` is the average percent error:

pmodelK = 1.0e+006 *

0.6083	-0.4055	0	0	0
-0.4055	0.5678	-0.1622	0	0
0	-0.1622	1.2437	-1.0814	0
0	0	-1.0814	1.6222	-0.5407
0	0	0	-0.5407	0.5407

calcK = 1.0e+006 *

0.5905	-0.4006	0	0	0
-0.4006	0.5560	-0.1554	0	0
0	-0.1554	1.2744	-1.1190	0
0	0	-1.1190	1.6580	-0.5391
0	0	0	-0.5391	0.5391

diffK = 1.0e+004 *

1.7782	-0.4968	0	0	0
-0.4968	1.1791	-0.6823	0	0
0	-0.6823	-3.0713	3.7535	0
0	0	3.7535	-3.5867	-0.1669
0	0	0	-0.1669	0.1669

percK = 2.1854

iterinfo = 9 -0.0000 0.0000 (last iteration, last fit improvement, norm of last search vector)

This is quite encouraging, as the changes to K shown in pmodelK are mirrored quite well in the calculated calcK, within about a 2% error. This was the case for many of the changes I tried, although most only took 2-5 iterations. The errors were all in the 2-3% range.

I have tested the algorithm with changes in m only, and have gotten similar results: 2-3% accuracy with around 2-5 iterations. This includes one interesting case where I modified m_4 by $+(m_5 * \frac{1}{4})$ and m_5 by $(m_5 * \frac{3}{4})$, simulating $\frac{1}{4}$ of the mass from the fifth floor falling on the fourth. I have also tested a variety of simultaneous changes in m and k , which requires finding all 10 parameters simultaneously. The accuracies are a little looser in this case, from 1.2 to 10% with an average of about 5.4%. The number of iterations can also increase dramatically, ranging from 8 to 72. This indicates that our predictor might not be as stable as it should. Indeed, I do get a warning about the predictor being unstable, so if this can be fixed the number of iterations should fall.

As a final test, Matt supplied 4 series of 5 outputs, each series representing a simple perturbation of the k values. My assignment was to determine how the k values had been changed. I was given the measured output, the input (white noise at the fifth floor as above), and the unperturbed model structure.

I ran FLAW_DETECTION using the default value of "tol" in FINDFLAW, 0.1. This determines the accuracy of the result as the iterations are continued until the search vector norm is less than "tol". The first round of results appear on the next-to-last stapled page.

For $\text{tol} = 0.1$, the results are interesting. Each perturbation required only a single iteration, indicating that perhaps tol should be lowered. The results indicated that:

Pert. 1: $k_1 \rightarrow k_1 * (0.5)$ (Actually 0.488)

Pert. 2: $k_3 \rightarrow k_3 * (0.5)$ (Actually 0.586)

Pert. 3: $k_5 \rightarrow k_5 * (0.5)$ (Actually 0.547)

Pert. 4: $k_1 \rightarrow k_1 * (0.8)$ (Actually 0.817)

$k_2 \rightarrow k_2 * (0.8)$ (Actually 0.821)

$k_3 \rightarrow k_3 * (0.8)$ (Actually 0.822)

$k_4 \rightarrow k_4 * (0.8)$ (Actually 0.824)

$k_5 \rightarrow k_5 * (0.8)$ (Actually 0.822)

To determine if greater accuracy was possible, I lowered tol to $1e-12$, effectively forcing the algorithm to search until it couldn't do any better. The results are on the last stapled page, and are:

Pert. 1: $k_1 \rightarrow k_1 \cdot (0.5)$ (Actually 0.504, 6 iterations)
 Pert. 2: $k_3 \rightarrow k_3 \cdot (0.5)$ (Actually 0.504, 13 iterations)
 Pert. 3: $k_5 \rightarrow k_5 \cdot (0.5)$ (Actually 0.498, 7 iterations)
 Pert. 4: $k_1 \rightarrow k_1 \cdot (0.8)$ (Actually 0.794, 9 iterations)
 $k_2 \rightarrow k_2 \cdot (0.8)$ (Actually 0.793)
 $k_3 \rightarrow k_3 \cdot (0.8)$ (Actually 0.792)
 $k_4 \rightarrow k_4 \cdot (0.8)$ (Actually 0.791)
 $k_5 \rightarrow k_5 \cdot (0.8)$ (Actually 0.805)

Matt has informed me that the above k identifications are correct. Thus for a single iteration we had a mean error of about 3.5%. For 5 to 12 more iterations, the mean error was down to about 0.5%.

Things to consider and to do next:

- 1) Do we want to use the diagonal Jordan form for F ? It diagonalizes F , at the cost of making G and H full. This will cause performance of higher order systems to suffer, while making the transformation of F much easier.
- 2) This procedure only works for full matrices. It will probably take considerable effort to change PEM so that it will take sparse matrices as arguments. I will have to dissect it and its subroutines to see.
- 3) The next step is probably to repeat these results on the 400 DOF system using 5 inputs and outputs as before. Then we can progress to the NTS structure.

INCLUDED FOR THE ELECTRONIC VERSION

Tol = 0.1

For perturbation 1:

deltak = 48.8626 -2.0639 -1.2526 1.1207 1.2829

calcK = 1.0e+6 *

2.4852	-1.6556	0	0	0
-1.6556	3.2981	-1.6425	0	0
0	-1.6425	3.2465	-1.6040	0
0	0	-1.6040	3.2053	-1.6014
0	0	0	-1.6014	1.6014

iterinfo = 1.0000 -0.0000 0.0000

For perturbation 2:

deltak = -6.7459 -6.2534 58.6045 -2.1101 1.7403

calcK = 1.0e+6 *

3.4552	-1.7236	0	0	0
-1.7236	2.3951	-0.6715	0	0
0	-0.6715	2.3279	-1.6564	0
0	0	-1.6564	3.2503	-1.5939
0	0	0	-1.5939	1.5939

iterinfo = 1.0000 -0.0000 0.0000

For perturbation 3:

deltak = 0.1973 0.3908 1.4524 4.4268 54.6924

calcK = 1.0e+6 *

3.2348	-1.6158	0	0	0
-1.6158	3.2144	-1.5986	0	0
0	-1.5986	3.1490	-1.5504	0

0 0 -1.5504 2.2853 -0.7350
0 0 0 -0.7350 0.7350

iterinfo = 1.0000 -0.0000 0.0000

For perturbation 4:

deltak = 18.3379 17.9310 17.8071 17.5667 17.8527

calcK = 1.0e+6 *

2.6560 -1.3313 0 0 0
-1.3313 2.6646 -1.3333 0 0
0 -1.3333 2.6705 -1.3372 0
0 0 -1.3372 2.6698 -1.3326
0 0 0 -1.3326 1.3326

iterinfo = 1.0000 -0.0000 0.0000

Tol = 1e-12

For perturbation 1:

deltak = 50.3705 0.8445 0.9643 1.1813 -0.5380

calcK = 1.0e+006 *

2.4135 -1.6085 0 0 0
-1.6085 3.2150 -1.6065 0 0
0 -1.6065 3.2095 -1.6030 0
0 0 -1.6030 3.2339 -1.6309
0 0 0 -1.6309 1.6309

iterinfo = 6.0000 0.0000 0.0000

University of California



For perturbation 2:

deltak = 0.6610 0.7923 50.3710 1.2274 -0.3737

calcK = 1.0e+006 *

3.2208	-1.6093	0	0	0
-1.6093	2.4144	-0.8051	0	0
0	-0.8051	2.4073	-1.6023	0
0	0	-1.6023	3.2305	-1.6282
0	0	0	-1.6282	1.6282

iterinfo = 13.0000 0.0000 0.0000

For perturbation 3:

deltak = 0.6875 0.7690 0.9768 0.8531 49.8414

calcK = 1.0e+006 *

3.2207	-1.6097	0	0	0
-1.6097	3.2160	-1.6063	0	0
0	-1.6063	3.2146	-1.6083	0
0	0	-1.6083	2.4220	-0.8137
0	0	0	-0.8137	0.8137

iterinfo = 7.0000 0.0000 0.0000

For perturbation 4:

deltak = 20.6029 20.6629 20.7924 20.8551 19.4538

calcK = 1.0e+006 *

2.5749	-1.2870	0	0	0
--------	---------	---	---	---

University of California

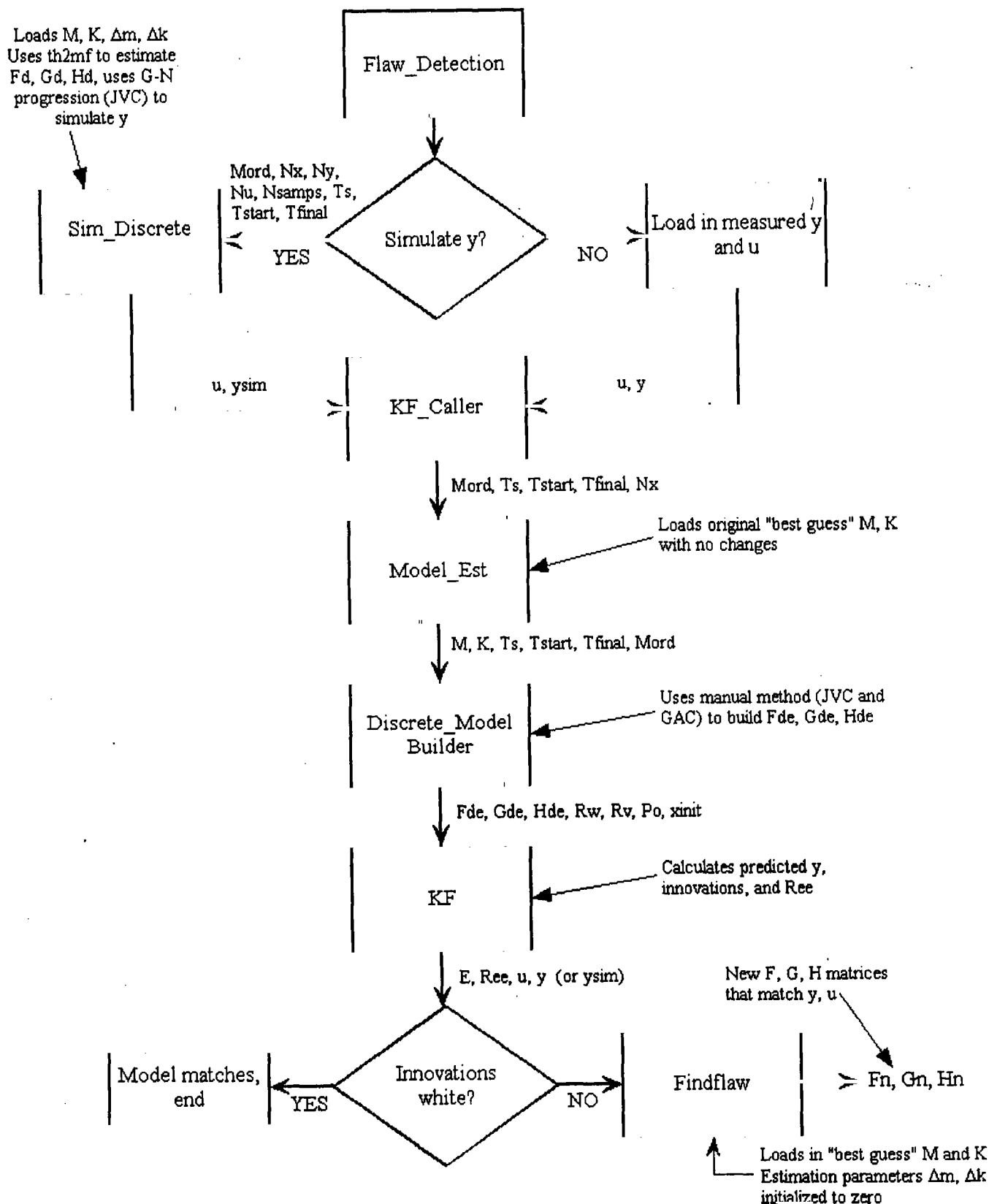


**Lawrence Livermore
National Laboratory**

-1.2870	2.5719	-1.2849	0	0
0	-1.2849	2.5687	-1.2839	0
0	0	-1.2839	2.5904	-1.3066
0	0	0	-1.3066	1.3066

iterinfo = 9.0000 0 0.0000

Flaw detection and identification algorithm



Interdepartmental letterhead
 Mail Station: L-271
 Ext: 33088

3/29/99

To: Greg Clark
 Jim Candy
 Dave McCallen
 Matt Hoehler

From: Greg Burnett

CC: Larry Ng

Re: Integration filter and results

This is a report on the filtering algorithm used to do the double integration that converts the noisy (drifting) accelerometer data from the NTS experiment. This data was marred by a long-term (low frequency) drift of the accelerometer voltage, which, when doubly integrated, resulted in a large distortion in the calculated displacement. This low frequency noise must be removed before the integration can be performed. Luckily, the experiment was set up so that only signals between 3 and 51 Hz can be considered to be actual data.

The indefinite integration from acceleration to velocity and then again to position of an arbitrary signal of frequency f proceeds as follows:

$$a = \sin(\omega t) = \sin(2\pi f t)$$

$$v = \int a \, dt = \frac{-\cos(2\pi f t)}{2\pi f} + v_0$$

$$x = \int v \, dt = \frac{-\sin(2\pi f t)}{(2\pi f)^2} + v_0 t + x_0$$

If we assume $v_0 = x_0 = 0$, then for any single frequency the ratio of the acceleration magnitude to the displacement will be $(2\pi f)^2$. This can be used to check the accuracy of the digital integration at several frequencies. It is important when doing so that the time vector begin with zero and not a finite number, otherwise the latter two terms in the last

equation come into play and distortion occurs. For the same reason a zero will be appended to the beginning of each data file to ensure stability.

The algorithm proceeds in the following manner:

- 1) The accelerometer data is read in and multiplied by 386.4 to translate English unit g's into inches.
- 2) The data is filtered with an 8th order Chebychev II HP filter that has a 3-dB frequency of 3 Hz and has the stopband at -60 dB. This filter is used because the Chebychev II filters have a maximally flat passband, eliminating any distortion in the passband. The noncausal Matlab command `FILTFILT` is used, which filters the data once with the filter, then reverses the data in time and filters it again, effectively doubling the order and eliminating any phase distortion. See figure 1 for the response of this filter.
- 3) The data is filtered with a 19th order Chebychev II HP filter that has a 3-dB frequency of 48.5 Hz and has the stopband at -60 dB. The `FILTFILT` command is also used. The response is shown in figure 2.
- 4) The resulting signal is integrated (to convert from acceleration to velocity) using a digital filter approximation to an analog integrator. The analog integrator response is simply

$$H_i(s) = \frac{1}{s}.$$

Using the bilinear transformation,

$$s \rightarrow \frac{2}{T} \left(\frac{z-1}{z+1} \right)$$

$$H_i(z) = \frac{1}{\frac{2}{T} \left(\frac{z-1}{z+1} \right)} = \frac{T}{2} \left(\frac{1+z^{-1}}{1-z^{-1}} \right)$$

or in Matlab format,

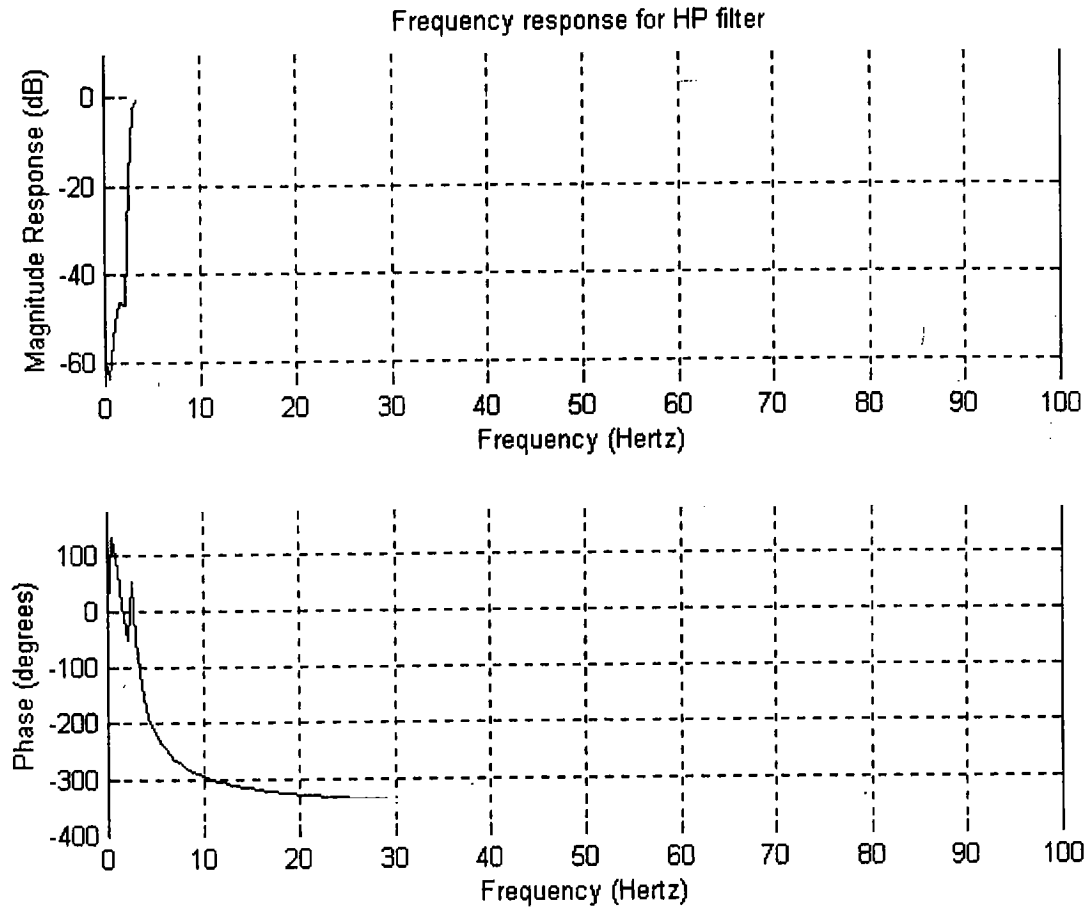


Figure 1. Frequency response for Cheb II HP filter with 3 dB frequency at 3 Hz.

$$B_i = \frac{T}{2} \begin{bmatrix} 1 & 1 \end{bmatrix}$$

$$A_i = \begin{bmatrix} 1 & -1 \end{bmatrix}$$

The frequency response for the digital integrator (red trace) is shown in figure 3 along with the response from the continuous-time integrator (red). The phase response for both is -90 degrees at all frequencies and is not shown. Note that the response of the digital integrator is excellent at low (< 30 Hz) frequencies. Indeed, at 30 Hz the difference is only 0.2 dB. At 40 Hz, the difference is only 0.3 dB, and at 50 Hz the difference is 0.45 dB. The fit gets rapidly worse above 100 Hz due to the presence of a zero in the transfer function at $z = -1$. This zero is absolutely necessary to get the required -90 phase shift at all frequencies, so the distortions in the magnitude

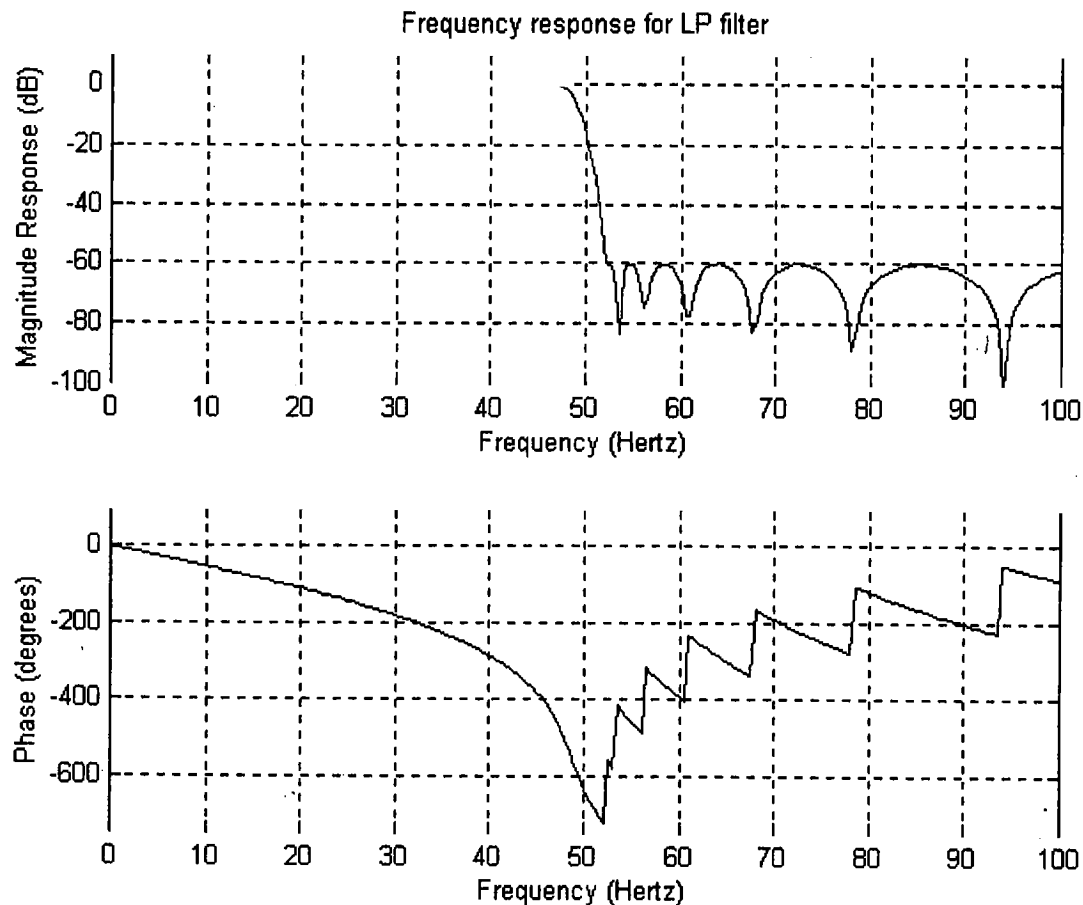


Figure 2. Frequency response for Cheb II LP filter with the 3-dB frequency at 48.5 Hz.

response cannot be removed. Care must be exercised so that the distortions due to the differences do not significantly affect the integration.

- 5) After integration, the data is again HP filtered using the filter shown in Figure 1. This is to remove the low frequency noise that integrating elevates to a significant level.
- 6) The integration and HP filtering are repeated for the change from velocity to displacement.

Testing the integration

To test the accuracy of the integration, I created sine waves ranging in frequency between 2 and 51 Hz and doubly integrated them using the filtering algorithm above. At each frequency the magnitude of the integrated signal was calculated using the method above and the results compared to the filtered signal. The errors at each frequency were

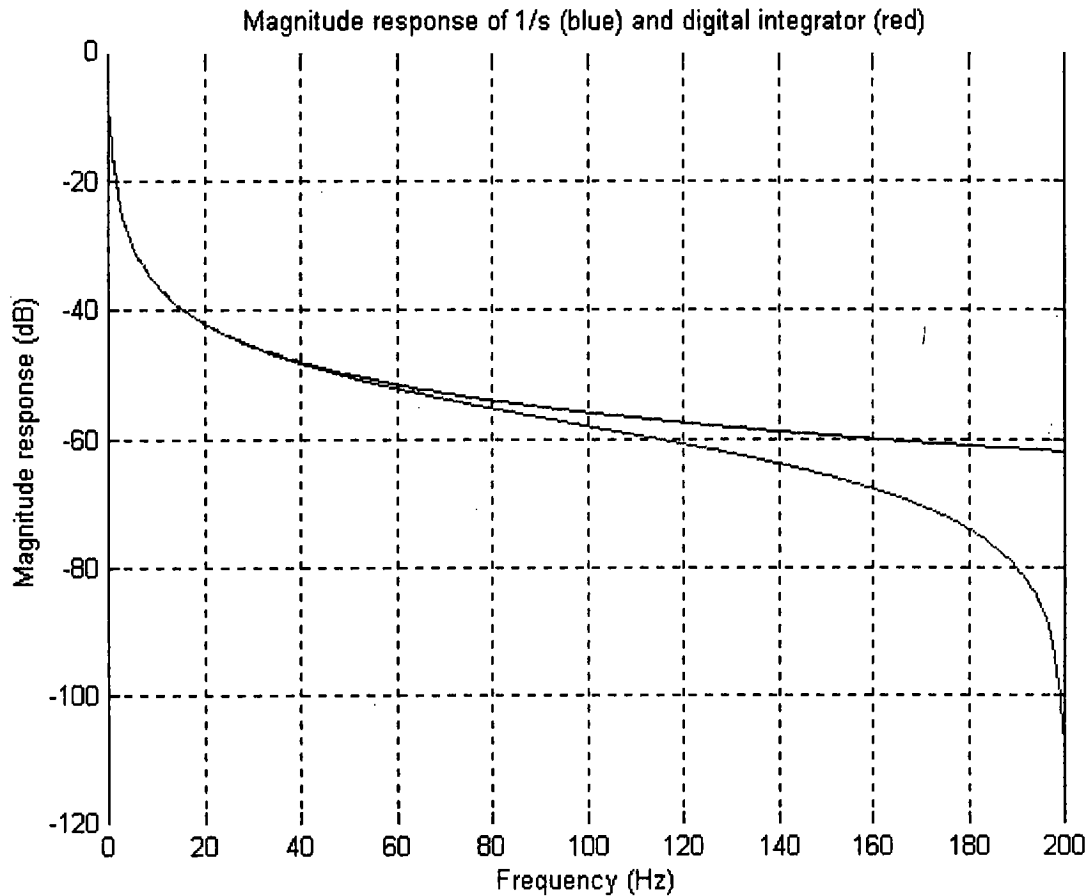


Figure 3. Comparison between the magnitude response for continuous time integrator (blue, upper) and a discrete time integrator (red, lower).

tabulated and are shown in Figure 4. The max error from 4 to 47 Hz is 10.1%, and the mean absolute error over the same range is 2.13%. Thus we may be fairly confident that the integration is accurate between 4 and 47 Hz.

One anomaly that has been observed is spurious 3 and 50 Hz signals introduced into the data by the filtering process. This is shown in Figure 5, where a 25 Hz sine has been used as the acceleration. The psd of the original data (top left) shows only the 25 Hz component. The psd of the signal after it has been HP and LP filtered shows two more frequency components at 3 and 50 Hz, which remain throughout the rest of the integrating process. It is not known at this time what is causing the extra frequencies, but they are normally more than 20 dB down and do not seem to appreciably distort the displacement signal.

Integration of representative data

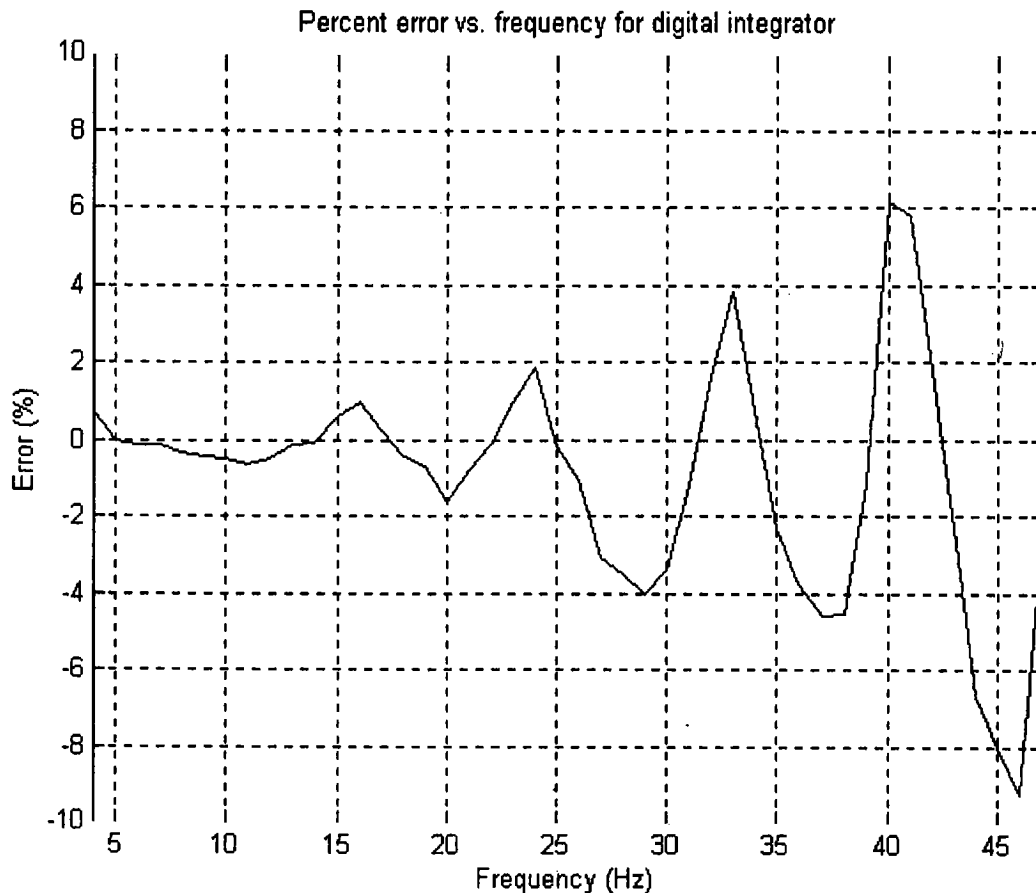


Figure 4. Percent error from 4-47 Hz for the digital integrator.

From the first swept experiment, `strucos1.mat`, channel 9 was recalled and used as the acceleration to the system. The displacement was generated using the integrating filter. The results are plotted in Figure 6, where the acceleration is on top (in g's) and the displacement on bottom (in inches). The magnitude of the displacement has been verified using the method above. The progress at each stage in the algorithm is shown in Figure 7.

Conclusions

The integration method described above may be used on data that contains information between 4 and 47 Hz with confidence. The magnitude of the doubly integrated signal matches closely to that expected from first principles. However, further research should be done to determine why extra frequencies are being introduced into the signals by the HP and LP filtering processes.

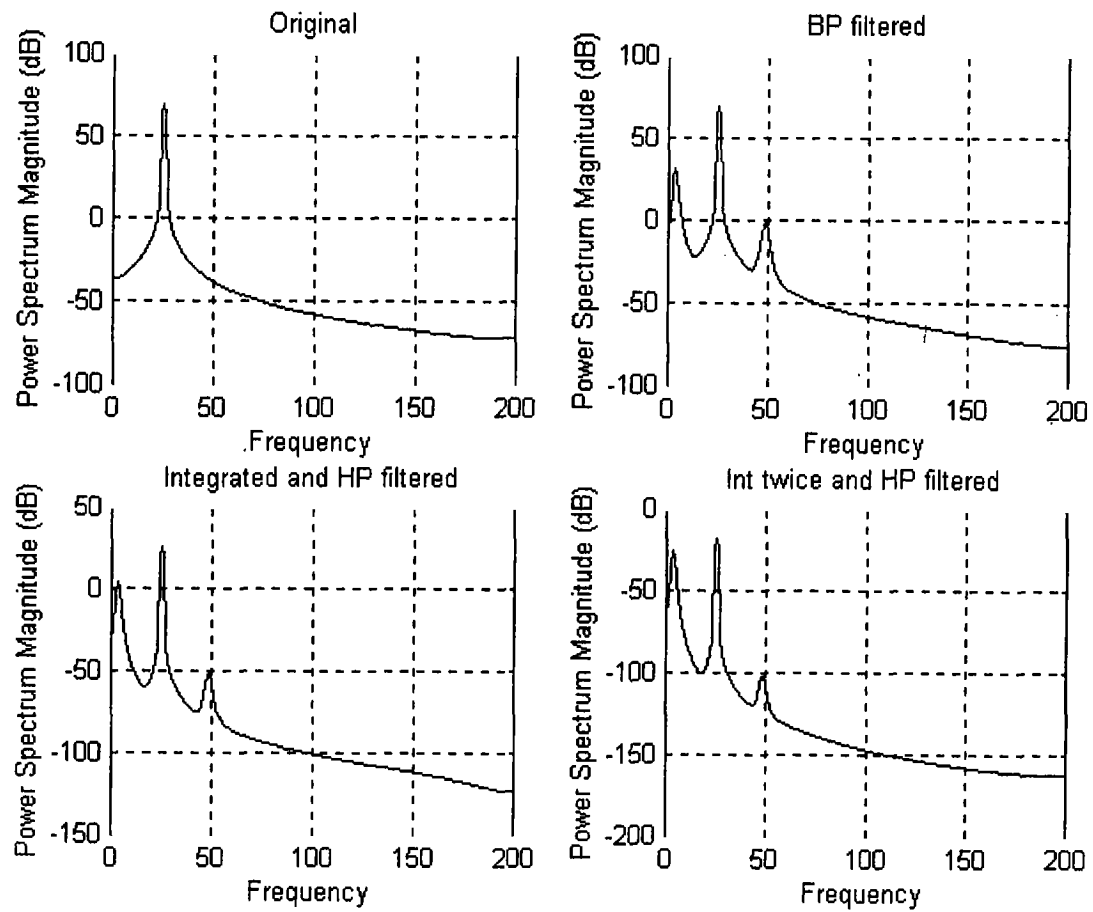


Figure 5. Progress along the integrating algorithm with a 25 Hz sine wave used as the acceleration signal. Upper left: the original signal. Upper right: the signal after HP and LP filters. Lower left: the signal after one integration and another HP filtering. Lower right: the signal after the second integration and HP filtering.

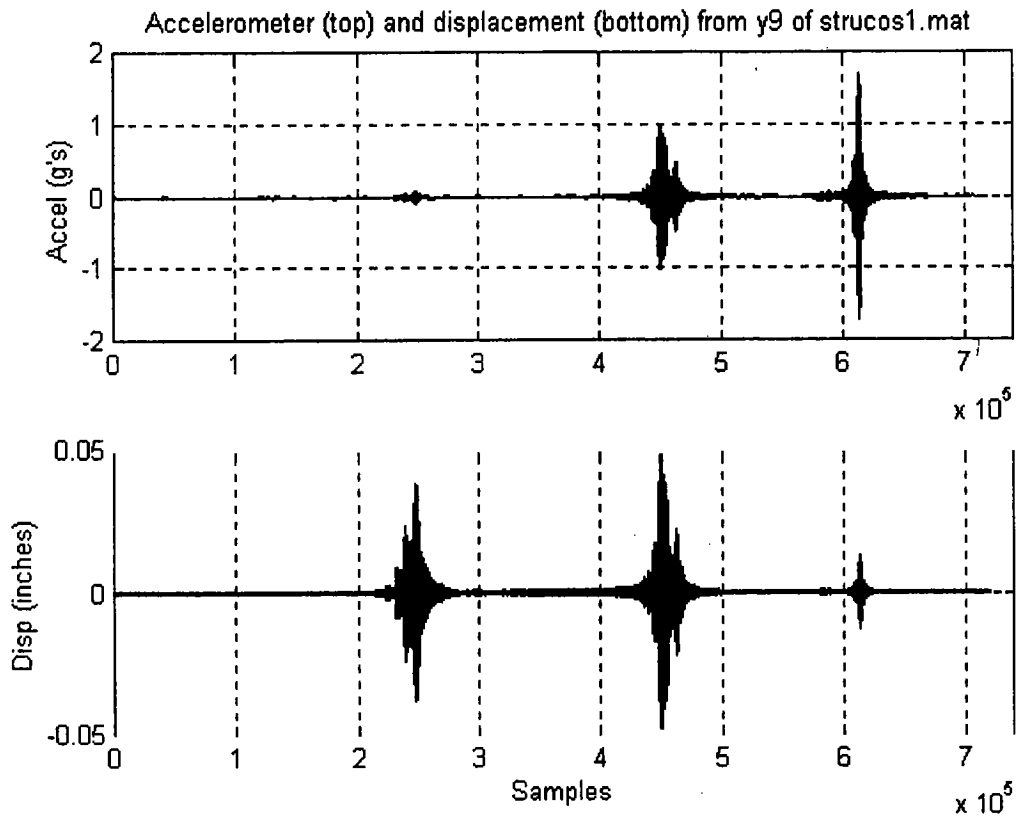


Figure 6. Acceleration (top) and calculated displacement (bottom).

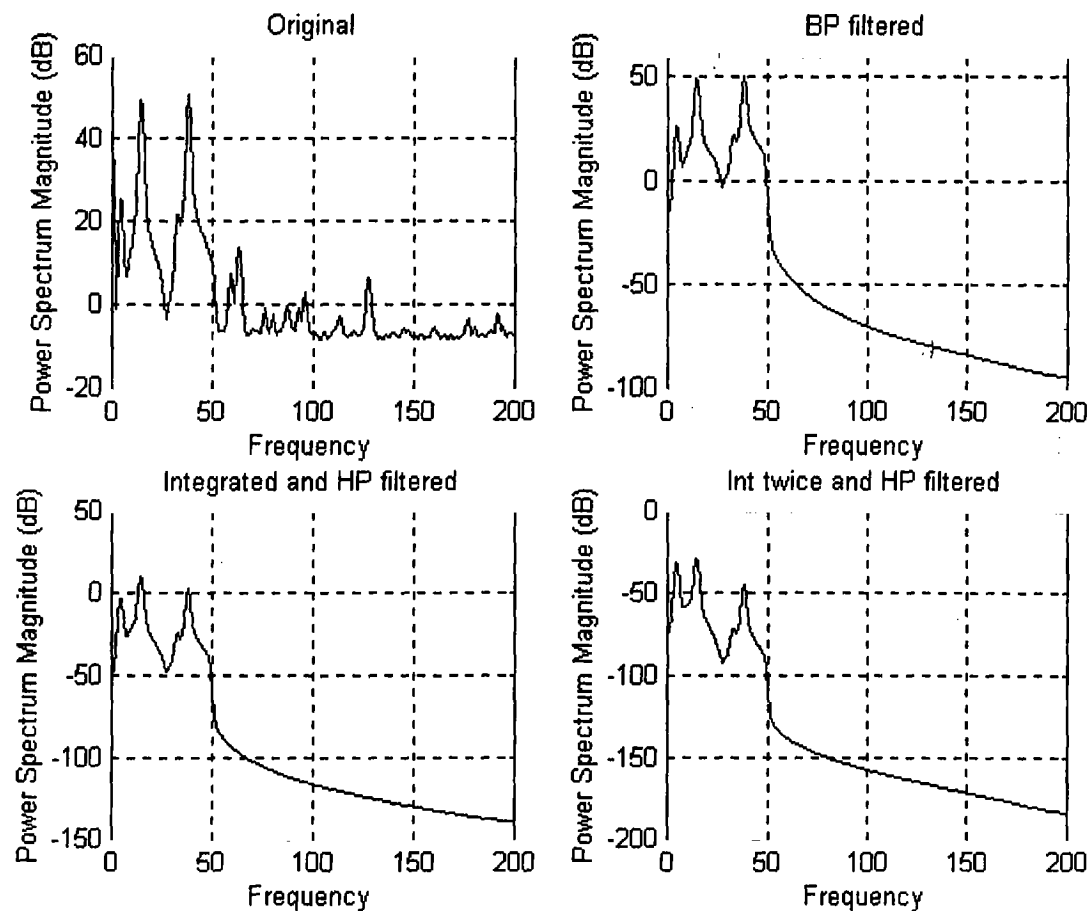


Figure 7. Progress along the integrating algorithm with an actual acceleration signal. Upper left: the original signal. Upper right: the signal after HP and LP filters. Lower left: the signal after one integration and another HP filtering. Lower right: the signal after the second integration and HP filtering.

System Identification

In the previous section, we discussed methods that can be used to identify the presence of damage in an established model or model mismatches of undamaged structures. Both situations result in non-white residuals, and can sometimes indicate the region where the damage has occurred or the model is lacking. Since the end result is the same (a model mismatch), the two situations can be considered to be variations of one another. Consequently, we will only treat model mismatch of undamaged structures in this text. The result is easily generalized for damage detection.

What is needed in finite element (FE) modeling is a systematic method of improving the model so that, through an iterative process, very accurate models can be constructed. At present, the educated guess of the modeler is the only improvement possible. Modelers can tweak parameters in an attempt to make the model more accurate, but this is a hit-and-miss process. There is no way to improve the FE model by using the measured response of the system to measured (or white) inputs. We are proposing a method based on FE modeling but implemented in a state-space environment using signal processing parameter estimation algorithms. This will give us the tools to perfect the FE models using measured inputs and outputs of a given physical system.

Parameter estimation

As the transition from FE modeling to state space and flaw detection has been explained in the previous section, we will proceed directly to parameter estimation. This is the process whereby a set of parameters in F , G , and H (arranged in vector form and known as the θ vector) are set as variables and estimated using iterative techniques. Not every physical constant in the system is thought of as variable. For instance, the length, width, and mass of a beam may be known to high precision. In this case, the variables are the strengths of the various types of connections between beams. They are all represented by various k_{ij} , where i and j represent the location of the k in a matrix that describes the interaction of the various elements at a discrete FE node location. In our five degree of freedom (DOF) example, there is only a single connection between each of the 6 nodes and thus the k matrix is just 1×1 .

The fit of the model to the experimentally derived data (consisting of i inputs and j outputs) is normally determined using the least squares criterion. As such, the innovation ϵ is defined as

$$\varepsilon = y - \hat{y} \quad \text{B.1}$$

and the minimization criterion V is

$$V(\theta, Z) = \frac{1}{N} \sum_{t=1}^N \frac{1}{2} \varepsilon(t, \theta)^2 \quad \text{B.2}$$

where N is the number of time samples.

The expression for the criterion V is then minimized with respect to θ . A fundamental iterative algorithm for finding the solutions of a function is the Newton method. In one dimension, this method can be visualized by plotting the function to be solved and then guessing one of the zero points. A more refined estimate is computed by drawing a line tangent to the function at our estimate x_c (c for current estimate) and determining where this line crosses the x axis. This distance is termed Δx , and the innovation is $\Delta y = y_c = f(x_c)$. Thus

$$\hat{x} = x_c - \Delta x, \quad \text{B.3}$$

where \hat{x} is the new estimate and

$$f'(x_c) = \frac{\Delta y}{\Delta x} = \frac{f(x_c)}{\Delta x} \quad \text{B.4}$$

so that

$$\hat{x} = x_c - \frac{f(x_c)}{f'(x_c)}.$$

Newton's method comes directly from Newton's theorem:

$$f(x) = f(x_c) + \int_{x_c}^x f'(z) dz$$

with the integral approximated by

$$\int_{x_c}^x f'(z) dz \cong f'(x_c)(x - x_c)$$

so that

$$f(x) = f(x_c) + f'(x_c)(x - x_c).$$

For a minimization problem, the function we want to solve is $f'(x)$, so Equation B.5 above becomes

$$\hat{x} = x_c - \frac{f'(x_c)}{f''(x_c)}$$

and for matrices this step becomes

$$\hat{x} = x_c - \frac{\mathbf{J}(x_c)}{\mathbf{H}(x_c)}$$

where \mathbf{J} is the Jacobian and \mathbf{H} is the Hessian of the matrix $\mathbf{F}(x_c)$ which we are trying to minimize.

In the least-squares problem, we want to minimize

$$f(\theta) = \frac{1}{2} \mathbf{E}(\theta)^T \mathbf{E}(\theta) = \frac{1}{2} \sum_{i=1}^m \epsilon_i(\theta)^2$$

where \mathbf{E} is the innovation function and $\epsilon_i(\theta)$ is the i^{th} innovation defined by Equation B.1 above.

The first derivative of $f(\theta)$ is defined by

$$\nabla f(\theta) = \frac{d}{d\theta} \frac{\mathbf{E}(\theta)^2}{2} = \mathbf{J}(\theta)^T \mathbf{E}(\theta)$$

Similarly, the second derivative of $f(\theta)$ is

$$\nabla^2 f(\theta) = \frac{d}{d\theta} \mathbf{J}(\theta)^T \mathbf{E}(\theta) = \mathbf{J}(\theta)^T \mathbf{J}(\theta) + \mathbf{H}(\theta)^T \mathbf{E}(\theta)$$

so that Newton's method appears as

$$\hat{x} = x_c - \frac{\mathbf{J}(\theta)^T \mathbf{E}(\theta)}{\mathbf{J}(\theta)^T \mathbf{J}(\theta) + \mathbf{H}(\theta)^T \mathbf{E}(\theta)}.$$

This method converges quite quickly if the initial guess is not too far off and there are no local minima nearby. Its drawback is that $\mathbf{H}(\theta)$ is quite expensive to obtain, and if the analytical form of $\mathbf{E}(\theta)$ is not available (as in our case) both \mathbf{J} and \mathbf{H} will have to be approximated using finite difference models or secant methods. This means on the order of n (where n is the model order) calculations for \mathbf{J} and $(3n^2 + n)/2$ calculations for \mathbf{H} .

In the Gauss-Newton method, the Hessian is discarded and the iteration proceeds as

$$\hat{x} = x_c - \frac{\mathbf{E}(\theta)}{\mathbf{J}(\theta)}.$$

The expense of the iteration is considerably reduced, but the performance of this method depends on the magnitude of $\mathbf{H}\mathbf{E}$ compared to $\mathbf{J}\mathbf{J}$. If $\mathbf{H}\mathbf{E}$ is much less than $\mathbf{J}\mathbf{J}$, this form closely approximates the pure Newton algorithm. This occurs when $\mathbf{E}(\theta)$ is linear in θ , or when the

innovation E is small. If these conditions are not met, the Gauss-Newton method may converge slowly or not at all.

Following are a list of advantages and disadvantages of the G-N method from Dennis and Schnabel p. 225 (1983):

Advantages:

1. Locally q -quadratically convergent on zero-residual problems.
2. Quickly locally q -linearly convergent on problems with small residuals or that are not too nonlinear.
3. Solves linear least-squares problems in one iteration.

Disadvantages

1. Slowly locally q -linearly convergent on problems with large residuals or that are sufficiently nonlinear.
2. Not locally convergent on problems with large or very nonlinear residuals.
3. Not well defined if J doesn't have full column rank.
4. Not necessarily globally convergent.

In our problem, especially for problems where the outputs are not fully observed, E and J will not have full column rank (i.e. there are less independent equations than the rank of the matrix) and this could cause difficulties. Also, if the model is sufficiently different from the actual structure the G-N algorithm might not converge at all.

To improve the convergence, the damped Gauss-Newton method was developed. It can be shown that the G-N method (Dennis and Schnabel (1983), p. 226) always takes steps in the correct direction, but sometimes these steps are too large, causing the method to diverge. The damped G-N algorithm simply takes the second term in Equation B.15 and multiplies by a constant of magnitude ≤ 1 :

$$\hat{x} = x_c - \lambda \frac{E(\theta)}{J(\theta)} = x_c - \lambda g$$

where g is defined as the Gauss-Newton search direction and λ is determined by one of several line-search methods (Dennis and Schnabel (1983), Chapter 6.3). One of the simplest (and that used in Matlab) is the halving line search, in which the criterion is tested for $\lambda = 1, 1/2, 1/4, \dots$ until a lower value for the criterion is found. This causes the damped G-N to be locally convergent for almost all nonlinear least-squares problems, including very nonlinear or large innovation problems. It is significantly more robust than the regular G-N algorithm. However, the convergence may be very slow.

These are important things to keep in mind as we examine how the damped G-N algorithm works in our state-space model.

The damped G-N algorithm in state-space

In our state-space model, the inputs $u(t)$ and outputs $y(t)$ in continuous time are related in terms of the states by

$$\begin{aligned}\dot{x}(t) &= \mathbf{F}x(t) + \mathbf{G}u(t) \\ y(t) &= \mathbf{C}x(t) + \mathbf{D}u(t) + v(t)\end{aligned}$$

and in discrete time by

$$\begin{aligned}x(n+1) &= \mathbf{A}x(n) + \mathbf{B}u(n) + w(n) \\ y(n) &= \mathbf{C}x(n) + \mathbf{D}u(n) + e(n)\end{aligned}$$

The model is first specified in continuous time (by using the equations of state for an oscillator, see section 1) and is transformed into discrete time by the following transformation:

$$\begin{aligned}\mathbf{B} &= \mathbf{\Psi} \mathbf{T} \mathbf{G} \\ \mathbf{A} &= \exp(\mathbf{F} \mathbf{T}) = \mathbf{I} + \mathbf{F} \mathbf{T} \mathbf{\Psi}\end{aligned}\tag{B.19}$$

where

$$\mathbf{\Psi} = \mathbf{I} + \frac{\mathbf{F} \mathbf{T}}{2!} + \frac{\mathbf{F}^2 \mathbf{T}^2}{3!} + \dots\tag{B.20}$$

which may be approximated to N terms for small $\mathbf{F} \mathbf{T}$ by

$$\mathbf{\Psi} \approx \mathbf{I} + \frac{\mathbf{F} \mathbf{T}}{2} \left(\mathbf{I} + \frac{\mathbf{F} \mathbf{T}}{3} \left(\dots \frac{\mathbf{F} \mathbf{T}}{N-1} \left(\mathbf{I} + \frac{\mathbf{F} \mathbf{T}}{N} \right) \right) \right),$$

(Franklin, Powell, Workmann, 1990, p.53) which is functionally identical to Equation B.24, but is possible to implement inexpensively. The only approximation is in the number of N terms used and for small $\mathbf{F} \mathbf{T}$, it is quite accurate. For large $\mathbf{F} \mathbf{T}$, however, this requires very large N (more than 200) as the higher order terms are quite large. A method for calculating $\exp(\mathbf{A})$ for large \mathbf{A} can be found in Kallstrom, 1973, and it is being requisitioned now so that this transformation can be done quickly and inexpensively (the Matlab program used takes about 30 minutes for a single transform on a fast computer). For small $\mathbf{F} \mathbf{T}$, the above transformation takes less than a second.

Once the discrete matrices are in hand, we can calculate the G-N search direction. From Equation B.16 we have:

$$g = \frac{\mathbf{J}^T(\theta)\mathbf{E}(\theta)}{\mathbf{J}^T(\theta)\mathbf{J}(\theta)} = \frac{\sum_{t=1}^N \mathbf{j}^T(n)\epsilon(n)}{\sum_{t=1}^N \mathbf{j}(n)\mathbf{j}^T(n)} \quad \text{B.22}$$

where again $\epsilon(n)$ is the innovation and $\mathbf{j}(n)$ is the gradient of the predictor $\hat{\mathbf{y}}$. It remains; then, to determine what these are in state space. In general, the state-space prediction models are given by (with \mathbf{D} assumed to be zero)

$$\begin{aligned} \hat{\mathbf{x}}(n+1 | \theta) &= \mathbf{A}(\theta)\hat{\mathbf{x}}(n, \theta) + \mathbf{B}(\theta)\mathbf{u}(n) + \mathbf{K}(\theta)\epsilon(n) \\ \hat{\mathbf{y}}(n | \theta) &= \mathbf{C}(\theta)\hat{\mathbf{x}}(n | \theta) \end{aligned} \quad \text{B.23}$$

where \mathbf{K} is the Kalman gain. These equations may be rewritten in a more convenient form as

$$\begin{aligned} \hat{\mathbf{x}}(n+1 | \theta) &= [\mathbf{A}(\theta) - \mathbf{K}(\theta)\mathbf{C}(\theta)]\hat{\mathbf{x}}(n, \theta) + [\mathbf{K}(\theta) \ \mathbf{B}(\theta)]\mathbf{z}(n) \\ \hat{\mathbf{y}}(n | \theta) &= \mathbf{C}(\theta)\hat{\mathbf{x}}(n | \theta) \end{aligned} \quad \text{B.24}$$

where

$$\mathbf{z}(n) = \begin{bmatrix} \mathbf{y}(n) \\ \mathbf{u}(n) \end{bmatrix} \quad \text{B.25}$$

and the innovation $\epsilon(n)$ has been written out explicitly as $(\mathbf{y}(n) - \mathbf{C}(\theta)\hat{\mathbf{x}}(n))$. This system of equations allow simpler processing by calculating a LTI response kernel based on

$$\mathbf{x}[n+1] = \tilde{\mathbf{A}}\mathbf{x}[n] + \tilde{\mathbf{B}}\tilde{\mathbf{u}}[n] \quad \text{B.26}$$

with $\tilde{\mathbf{A}} = [\mathbf{A} - \mathbf{K}\mathbf{C}]$, $\tilde{\mathbf{B}} = [\mathbf{K} \ \mathbf{B}]$, and $\tilde{\mathbf{u}}[n] = \mathbf{z}[n]$. A built-in Matlab function (`ltitr.m`) is used to calculate the response if full matrices are used, for sparse matrices I constructed a similar function named `ltitr_sparse.m`.

To find the gradient we differentiate the predictor $\hat{\mathbf{y}}$ in B.24 w.r.t. θ :

$$\psi(n | \theta) = \frac{d(\hat{\mathbf{y}}(n | \theta))}{d\theta} = \mathbf{C}(\theta)\Psi_x(n | \theta) + \mathbf{C}'\mathbf{x}(n | \theta) \quad \text{B.27}$$

where Ψ_x is the derivative of $\hat{\mathbf{x}}$ w.r.t. θ and \mathbf{C}' is the derivative of \mathbf{C} w.r.t. θ . The quantity Ψ_x can be expressed as the derivative of B.24 (with the θ argument suppressed)

$$\Psi_x(n+1) = (\mathbf{A} - \mathbf{K}\mathbf{C})\Psi_x(n) + (\mathbf{A}' - \mathbf{K}'\mathbf{C} - \mathbf{K}\mathbf{C}')\hat{\mathbf{x}}(n) + \mathbf{K}'\mathbf{y}(n) + \mathbf{B}'\mathbf{u}(n) \quad \text{B.28}$$

The G-N direction g can now be calculated by forming the matrix

$$\Psi = [\psi(1), \dots, \psi(i)]^T \quad \text{B.29}$$

where there are i indices (variables of the problem), and the matrices Ψ have been vectorized row by row, and the vector

$$E = [\epsilon(1) \dots \epsilon(k)]^T \quad \text{B.30}$$

(where k is the number of members of ϵ) and solving for g in an overdetermined linear equation using the LS technique (Ljung p.302) B.31

$$\Psi g = E$$

Now that we have g for our set of parameters, we can search along the g direction, looking for a lower value of the criterion V . Using the damped G-N method above, we have

$$\theta = \theta_0 + \lambda g^T \quad \text{B.32}$$

where θ_0 is the old theta parameters and λ is the line search constant. It is varied (as stated above) only if a lower value of the criterion cannot be found. That is, λ starts out as 1. The parameters A , B , C , K , and X_0 are calculated from the given model using the new θ . The innovation and criterion are calculated, and if the criterion is not lower, it is assumed the algorithm has overshoot and the process is repeated with $\lambda = 1/2$. This continues until a lower value is found or until $\lambda = \frac{1}{2^9}$. At this point the algorithm terminates.

If a lower value of V is found, the algorithm checks to see if V is below the termination threshold. If it is not, the new θ vector is used to calculate the innovation, predictor, and gradient and a new search direction is calculated. The process repeats until V is less than the specified threshold. Then the most current θ is returned as the solution to the problem.

We have not discussed observability yet, which will be addressed in the next memo, along with algorithms designed especially for sparse systems such as ours.

References:

1. J.E. Dennis and R.B. Schnabel (1983), chapter 10. "Numerical Methods for Unconstrained Optimization and Nonlinear Equations", Prentice-Hall, Englewood Cliffs, NJ. ISBN 0-13-627216-9

2. Lennart Ljung (1987). "System Identification theory for the user". Prentice-Hall, Englewood Cliffs, NJ. ISBN 0-13-881640-9
3. Franklin, G.F., Powell, J.D., Workman, M.L. (1990). "Digital control of dynamic systems". Addison-Wesley Publishing Company, Reading, MA. ISBN 0-201-11938-2
4. Kallstrom, C. (1973). "Computing $\text{EXP}(A)$ and the integral of $\text{EXP}(As)$ ds". Report 7309, Lund Institute of Technology, Division of Automatic Control, March 1973. *Not yet located.*

Interdepartmental letterhead
Mail Station: L-271
Ext: 33088

6/2/99

To: Greg Clark, Jim Candy
 From: Greg Burnett
 CC: Dave McCallen
 Re: Continuous to discrete transformations

Hello Greg and Jim,

I have run into a problem while scaling the parametric identification algorithm from the 5 DOF I wrote about in my memo of March 15th to 400 DOF. The problem is that there are 1812 variables in the theta vector, so for each iteration in the identification process the state-space matrices **A**, **B**, **C**, and **D** have to be reformulated from the perturbed theta vector and **M**, **C** and **K** 1812 times. Naturally, I would like this to be a quick process. I have improved the time and memory requirements substantially by converting all processes (modeling, simulation, and flaw detection) to work with sparse matrices, but there is one thing I am having a lot of trouble with (and I'm not sure if it's a real problem or if I've just looked at it so long I've lost sight of an obvious solution). Thus this memo.

The problem is the transformation of the state space matrices from continuous to discrete space. Specifically, when transforming **F** and **G** to **A** and **B** we use (from Franklin, Powell, and Workman, p. 53)

$$\mathbf{A} = e^{\mathbf{F}T}$$

$$\mathbf{B} = \int_0^T e^{\mathbf{F}t} dt \cdot \mathbf{G}$$

The matrix **A** may be approximated by a Taylor series expansion:

$$\mathbf{A} = e^{\mathbf{F}T} = \mathbf{I} + \mathbf{F}T + \frac{\mathbf{F}^2 T^2}{2!} + \frac{\mathbf{F}^3 T^3}{3!} + \dots$$

which can also be written as

C.1

$$\mathbf{A} = \mathbf{I} + \mathbf{FT}\Psi$$

with

$$\Psi = \mathbf{I} + \frac{\mathbf{FT}}{2!} + \frac{\mathbf{F}^2\mathbf{T}^2}{3!} + \dots$$

The \mathbf{B} integral above may be integrated term by term to give

$$\mathbf{B} = \sum_{k=0}^{\infty} \frac{\mathbf{F}^k \mathbf{T}^k}{(k+1)!} \mathbf{TG} = \Psi \mathbf{TG} \quad \text{C.2}$$

To be more computationally efficient, FP&W suggest evaluating Ψ using the form

$$\Psi \approx \mathbf{I} + \frac{\mathbf{FT}}{2} \left(\mathbf{I} + \frac{\mathbf{FT}}{3} \left(\mathbf{I} + \frac{\mathbf{FT}}{4} \left(\dots \frac{\mathbf{FT}}{N-1} \left(\mathbf{I} + \frac{\mathbf{FT}}{N} \right) \right) \right) \right)$$

which allows an iterative approach with better numerical properties. I used it in the 5 DOF case, using $N = 15$, and the calculation was quite rapid and accurate. However, it falters in the 400 DOF case, in which it needs up to 200 terms to have accuracy to within a few percent. This is due to the large values of \mathbf{F} , in which the 2-norm is 2.9×10^{10} . An obscure reference is cited (Kallstrom 1973) on how to approximate the exponential for large \mathbf{FT} , but it has not yet been located.

At this time I consulted a reference suggested by Jim, "Nineteen dubious ways to compute the exponential of a matrix" by Moler and Van Loan (1978). Most of the more accurate methods require the eigenvalues of \mathbf{A} , and that is a very expensive (~45 seconds) calculation for this problem, so they were discarded. The methods that drew my interest the most were Method 3 (scaling and squaring) and Method 19 (splitting).

Method 3 involves the use of the property of the exponential:

$$e^{\mathbf{A}} = (e^{\mathbf{A}/m})^m$$

The idea is to choose a power of two for m such that $e^{\mathbf{A}/m}$ is easily calculated using Taylor or Pade approximations and then to reform $e^{\mathbf{A}}$ by repeated squaring. The commonly used criterion for choosing m is to make it the smallest power of two so that

$$\frac{\|\mathbf{A}\|}{m} \leq 1$$



where $\|A\|$ is the 2-norm

$$\|A\| = \left(\sum_{i=1}^n |x_i|^2 \right)^{1/2}.$$

Once e^{AT} is calculated, B can be calculated by solving for Ψ in Equation C.1 and substituting it into Equation C.2:

$$\Psi = \frac{A - I}{FT}$$

$$B = \left(\frac{A - I}{F} \right) G$$

This can lead to trouble if F is ill-conditioned, where the condition is $\|F\| \cdot \|F^{-1}\|$. If the condition is large, then F is almost singular and is ill-conditioned.

To test the accuracy and speed of this algorithm, I ran several experiments using identity matrices A that were multiplied by different positive constants C to yield a variety of sizes and 2-norms. I compared the accuracy and time required for the Matlab algorithm EXPM.M and the sparse implementation of Method 3 using $N = 5$ (only the first five terms of the Fourier approximation were used). The value returned by EXPM.M was defined as the “real” answer for the error calculation. Table 1 lists the matrices and their average error:

A size	C	m	T _{sparse}	T _{expm}	T _B	% error
1 x 1	1	0	0	0	0	0.06
1 x 1	100	7	0.01	0	0	2.06
1 x 1	500	9	0	0	0	23.6
1 x 1	1000	10	N/A	N/A	N/A	Inf
5 x 5	1	2	0	0	0	0.00011
5 x 5	100	8	0.01	0	0	0.09
5 x 5	500	11	0.01	0	0	0.05
100x100	1	4	0.01	0.13	0.01	0
100x100	100	10	0.03	0.23	0	1.1e-4

100x100	500	13	0.04	0.25	0	5.6e-5
500x500	1	5	0.11	31.93	0.01	0
500x500	100	12	0.15	54.53	0	0
500x500	500	14	0.16	62.07	0	1.8e-6
500x500*	1	6	0.65	38.21	1.40	3.1e-5
500x500*	100	12	0.82	61.7	1.30	5.0e-4
500x500*	500	14	0.90	70.04	1.30	5.0e-4

where A size is the size of A , C is the constant that multiplies A , m is the factor of two used, T_{sparse} is the time for the sparse method, T_{exp} is the time for Matlab's built-in matrix exponential, T_B is the time to calculate B using A , and % diff is the relative percent difference (which is reported as zero if below 1×10^{-6}) between the results of the scale and square sparse algorithm and $\text{expm}(CA)$. All times are in seconds. A result of N/A means the number was too large for Matlab to measure. All matrices were simple identity matrices except for the last three marked with an *, which included four off-diagonals to simulate the sparsity of the real F and G matrices. For these experiments, G was taken to be equal to F . As an example, the sixth experiment used a 5×5 identity matrix multiplied by 100. The sparse Method 3 took 8 squarings to reach the desired accuracy, and each process took 10 milliseconds or less to complete. The relative error between the two algorithms was 0.09%.

It is clear that the sparse algorithm is much faster for the larger matrices and quite accurate, except for the scalar multiple of 500. This is to be expected because 5 components of the Taylor series will not be enough for such large values of A . For the large matrices, though, accuracy and speed are quite acceptable. However, there is a problem that is not evident from these simple experiments.

The problem lies in the squaring loop. In the process of squaring, roundoff errors accumulate and gradually fill the sparse matrices with very tiny numbers, slowing performance substantially. To remedy this, at each iteration in the squaring process A is flushed of all values below a threshold, which is determined by taking the average of the absolute value of the largest $3n$ values, where n is the model order and three is used to capture the main diagonal and two off-diagonals. This process works well for the example above, in which the test matrices' components are all of the same order. However, in our

problem **F** has members that vary on the order of 10^6 . This makes the calculation of the flushing threshold quite difficult. Too high a threshold and the smaller values are removed, leading to an inaccurate **A** and **B**, and too low a threshold leads in many spurious nonzero values, slowing the transformation considerably. To attempt to sidestep this difficulty we examine the next method.

Method 19 is called the splitting technique. Here, e^A is approximated by splitting **A** into **B** + **C** and using the approximation

$$e^A \cong (e^{B/m} \cdot e^{C/m})^m$$

in which the equality is true only if **B** and **C** commute, that is $[B, C] = BC - CB = 0$. The advantages of this algorithm are that if the scale of **B** and **C** vary substantially, each can be calculated separately using Method 3 with different thresholds. The factor m can be determined using the inequality

$$\|e^A - (e^{B/m} \cdot e^{C/m})^m\| \leq \frac{\|[B, C]\|}{2m} e^{\|B\| + \|C\|}.$$

Since our **F** consists of two distinctly scaled matrices, we could split it and try to compute the individual exponentials:

$$[F] = \begin{bmatrix} 0_N & I_N \\ -\frac{K}{M} & -\frac{C}{M} \end{bmatrix} = \begin{bmatrix} 0_N & I_N \\ 0_N & 0_N \end{bmatrix} + \begin{bmatrix} 0_N & 0_N \\ -\frac{K}{M} & -\frac{C}{M} \end{bmatrix}$$

To estimate the error, we calculate $[B, C]$:

$$[B, C] = \begin{bmatrix} -\frac{K}{M} & -\frac{C}{M} \\ 0_N & 0_N \end{bmatrix} - \begin{bmatrix} 0_N & 0_N \\ 0_N & -\frac{K}{M} \end{bmatrix} = \begin{bmatrix} -\frac{K}{M} & -\frac{C}{M} \\ 0_N & \frac{K}{M} \end{bmatrix}$$

Now the norms of $[B, C]$ and **B** and **C** may be calculated. The norm of $[B, C]$ is 2.87×10^3 , and the norms of **B** and **C** are 2.24 and 1.65×10^3 , respectively. Neglecting the norm of **B** and assuming a generous error norm of 1000, m would still have to be very large due to the large norm of **C**. In fact, m would have to be greater than 1×10^{300} , not too useful for our (or anybody else's) calculations. It might be possible to factorize **F** into better matrices, but we would still have the problems of scale associated with the problem that we experienced earlier. Thus this method will not be very useful for our calculations.

So that is where we stand. The scaling and squaring method works well but the threshold factor can cause the answer to be incorrect or take forever to calculate. The splitting method will not work because the norm of \mathbf{C} is too high. So what are our options?

Scaling is not an option. Right now, the units of \mathbf{M} are in $\text{lb} \cdot \text{feet} \cdot \text{seconds}^2 / \text{inch}$ and \mathbf{K} is in $\text{lb} \cdot \text{feet} / \text{inch}$. If we change these to units of 1000 lb and 1000 feet, then the base unit of \mathbf{M} would change from 5045 to 5.045×10^{-3} and the base of \mathbf{K} would change from 1.62×10^6 to 1.62. Thus our scaling problem is seemingly lessened by a factor of 10^3 . However, the matrix of interest (\mathbf{F}) is constructed from the ratio of \mathbf{K} to \mathbf{M} , which is unaffected by scaling. Thus, scaling is not helpful.

At this point I am simply going to try and find a threshold that is both reasonably accurate and fast. If either of you have any suggestions or references, please send them to me when you get a chance.

Thanks!

Greg

References:

Franklin, G.F, Powell, J.D., Workman, M.L. (1990). *Digital control of dynamic systems*, Addison-Wesley, Reading MA. ISBN 0-201-11938-2.

Kallstrom, C. (1973). "Computing $\text{EXP}(\mathbf{A})$ and the integral of $\text{EXP}(\mathbf{A}s)ds$ ", Report 7309, Lund Institute of Technology, Division of Automatic Control, March 1973. *Not yet located*.

Moler, C. and Van Loan, C. (1978). "Nineteen dubious ways to calculate the exponential of a matrix", SIAM Review, Vol. 20 (No. 4), October 1978, p.801-836.

Interdepartmental letterhead
Mail Station: L-271
Ext: 33088

7/7/99

To: Greg Clark
Jim Candy
Dave McCallen
Matt Hoehler

From: Greg Burnett

CC: DRAFT

Re: 50 and 400 DOF progress

Hello everyone,

I wanted to write a memo on the progress and pain of simulating and identifying the 400 DOF system. At present, I am still having numerical problems. I believe it is probably due to the observability (or lack thereof) of the system.

Background:

The parameter identification method used for the 400 DOF problem is a modified version of PEM, written by Lennart Ljung. It utilizes several subroutines, and utilizes the damped Gauss-Newton algorithm described in my draft report to Greg. The program, as written, does not work on sparse matrices at all. It utilizes several built-in functions (such as EIG) that crash when given sparse matrices. Since our matrices are by nature quite large (from 400 by 400 to 28800 by 215 for the 400 DOF problem), it was necessary to rewrite PEM and its associated subroutines to operate sparsely. At the same time, the 4 major subroutines were merged together for greater speed and efficiency. The result was PEM_SPARSE. So up to this point the progress has been:

1. Rewrite PEM, PEMSS, SESS, GNSS, MF2TH, LTITR and C2D to accept sparse matrices.
2. Incorporate PEM, PEMSS, SESS, and GNSS into a single new program, PEM_SPARSE.
3. Test on the 5 DOF system.

University of California



**Lawrence Livermore
National Laboratory**

At this point (late April) the algorithm functioned well for 5 DOF. It was a little slower than PEM since the 5 DOF system is not very sparse, but it gives the same answers. I looked into compiling the major subroutines (those that have to run many times per iteration, LTITR_SPARSE and C2DGB), but the Matlab compiler will not compile a routine that utilizes sparse matrices. Thus we can either use full matrices with compiled functions and take a huge memory hit (it balloons into many hundreds of MB) or use uncompiled sparse functions, which only require about 10 MB of memory but are slow. I have recently located a third-party compiler which claims to compile sparse functions, but have not yet had a chance to try it.

I now began to test the algorithm with 400 DOF. It was necessary to write a program that would assemble the discrete **F**, **G**, and **H** matrices needed for simulation and identification given the unknowns of the system. At this point we decided to use all of the nonzero diagonal and upper triangle values in the **K** matrix as our unknowns (all 1812 of them). By determining which components of the **K** matrix were incorrect, we could determine which elements might have been damaged. This process must be done many times per iteration, and time was saved by only updating the damping constants (the first and fourth eigenvalues of $\mathbf{M}^{-1}\mathbf{K}$ every hundredth iteration. This will substantially speed up the process without sacrificing too much accuracy.

4. Write TH400DOF, which assembles the **M**, **C**, and **K** matrices given the original values of **K** and the (proposed) changes to it.

The 400 DOF algorithm was very slow at first, so I spent a good deal of time optimizing LTITR_SPARSE and C2DGB. LTITR is basically a linear time-invariant kernel propagator, which calculates the response of $\mathbf{x}[n+1] = \mathbf{A}\mathbf{x}[n] + \mathbf{B}\mathbf{u}[n]$ given **A**, **B**, **u**, and $\mathbf{x}[0]$. It is used extensively to calculate the innovations and gradient direction. C2DGB is the sparse variant of C2D, which samples the continuous system described by **A**, **B**, and **C** and returns the discrete matrices **F**, **G**, and **H**. It is necessary to convert the matrices in order to compare the simulated outputs with the actual sampled outputs of the system. The algorithm for this is contained in my memo of 6/2/99, entitled *Continuous to discrete transformations*. I had some difficulty getting the algorithm to behave, but with a good amount



of help from Jim and Greg I have gotten it to operate quite smoothly. Although the poor scaling slows down the calculation, it now seems to be quite accurate and stable.

5. Optimize LTITR_SPARSE and C2DGB for speed and poorly scaled matrices.

Just before I left on vacation in the middle of June, we decided to change the way the 400 DOF system matrices are constructed. Instead of using the 1812 members of \mathbf{K} , we decided to use the E values in each of the 215 elements. Thus any differences in the modeled and identified matrices could be traced to a single element. This decreases the size of the algorithmic matrices substantially and speeds calculations to where a single iteration takes only a few hours, not on the order of a day. I wrote TH400DOF_SMALL_K to assemble the matrices given the values of E for each element, and assumed that each floor node could be measured. This gives 20 measurements of the 400 equations.

6. Rewrite TH400DOF so that the individual beam and floor E are the variables of the system.
7. Assume each floor node is measured.

As it stands now, the problem I am having is that the Ψ matrix is singular. To review, the direction of the iterative improvement g is given by

$$g = \frac{\mathbf{E}}{\Psi}$$

As my Ψ is singular, the algorithm fails. I believe that Ψ is singular because the problem, as it is set up, is not observable. I would like to try a smaller system with only a single discretized column for a total of 50 DOF. Matt has given me the matrices; I just need to build a TH50DOF_SMALL_K so that I can do the identification. I could even try it with every node measured, just to see if observability is a problem. I don't think this is practical with the 400 DOF system as it would no longer be sparse and would take too long to do.

Update 7/15/99

I have received and implemented pem_sparse for both the 10 and 50 DOF systems. The results for the 10 DOF system are excellent: with 3-6 iterations, it is accurate to within a few percent even for very large (up to a 90% change tested) perturbations of the EI values. In order to get the algorithm to work when every DOF is not observed, I had to “collapse” certain matrices (such as the innovations) so that other matrices (such as the gradient) wouldn’t become singular. By collapsing key matrices, I could keep the calculations regular and speed up the calculations due to the smaller matrix size.

In addition to this, I found it necessary to restrict the parameters used in calculating the Gauss-Newton search direction to ensure stability in the criterion calculation. The algorithm right now sometimes calculates G-N directions g that have members 100 times larger than the parameters themselves. This can cause the EI values to become negative, which is nonphysical and results in errors. I restrict the new parameters calculated by

$$\text{newpars} = \text{oldpars} + k \cdot g$$

to values between $\text{oldpars} / 100$ (severe damage) and $2 * \text{oldpars}$ (model mismatch). This makes the algorithm much more stable. In the future it might be useful to limit the size of the members of g as well.

The 50 DOF, as of 2:07 this afternoon, is still not converging to the correct answer using the ME simulator (ours is returning values for y that are ~1000 times too large). Right now we are perturbing the EI parameters EI(6):EI(10) by ½ of their original value. Chad is preparing a second series, which will have all translational nodes observed, and I will work on that later today to see if it’s an observability problem.

Interdepartmental letterhead
Mail Station: L-271
Ext: 33088

9/17/99

To: Dave McCallen
Greg Clark
Larry Ng
Chad Noble
Todd Gable

From: Greg Burnett

CC:

Re: The State of the Art in Vibration-Based Structural Damage Identification

Hello everyone,

I wanted to write a quick memo and summarize my thoughts on the tutorial that Chad and I attended earlier this week. Here are some quick impressions:

- 1) The presenters are well versed in the state of the art of vibrational analysis. They were quite knowledgeable and presented the material in an organized, efficient manner. This tutorial focused on damage identification, but they also do courses on FEM update routines. I feel it would be worthwhile to invite them here for a short course on model updating so we can get a feel for the current state of the art in this area.
- 2) The Kalman Filter-based residual whiteness test conceived by Jim and Greg is as good or better than anything currently being used for damage detection. The most advanced system they offered at the tutorial was an LPC-based residual whiteness analysis. The Kalman Filter approach is more robust, noise-tolerant, and accurate. Therefore our damage detection algorithm is among the best out there. It needs to be tested on real data, which I will accomplish soon.
- 3) There is quite a bit of literature on model update algorithms, but most of them seem to be concentrated on modal analysis methods. I am going to research the references they recommend and see if there is anything useful for us there.
- 4) We should test the real structures using a coherence test to see how nonlinear the structure is. If it is significantly nonlinear, we may have trouble fitting the model to the data.

University of California



**Lawrence Livermore
National Laboratory**

- 5) We could use the Principle Component Method on the recorded data to determine what nodes are more important than others. This would also allow us to redistribute sensors on a structure after we have taken data so that more important locations are covered in more detail.
- 6) A two step process is commonly used to identify damage in large structures that would otherwise be computationally perverse. The first identifies the general area of damage and the second concentrates on the general area with a more detailed algorithm to more closely identify the damage.
- 7) Genetic algorithms might possibly be useful for large DOF systems in which many local minima are present. They have a slightly tarnished reputation, though, as they have been misused in the past.

GEMS relevance

- 8) There are techniques that can be used to determine confidence levels on calculated transfer functions. We could use those when the transfer functions are used as a metric in speaker identification.
- 9) The "bootstrap" procedure could be used by Todd in his classification process to calculate the properties of his template.
- 10) The Fischer discriminant can be very useful in separating data that are caused by different underlying processes. It would be very useful for Todd when he is classifying speakers.

That's about it for now. If any of you have any comments or would like more details, please let me know.

Greg



Interdepartmental letterhead
 Mail Station: L-271
 Ext: 33088

10/18/99

To: Greg Clark
 Larry Ng
 Dave McCallen
 Chad Noble

From: Greg Burnett

CC:

Re: The 5 DOF NTS model using the damped Gauss-Newton algorithm

Hello everyone,

I wanted to take the time to write out the progress of the 5 DOF NTS structure. I have been trying several different approaches, and with Larry's help have made some significant progress. However, I have not been able to improve the model using the data we captured at the NTS. Hopefully this memo will explain why and how we might go about solving the problem.

Review of the algorithm

This is a summary of the damped G-N algorithm which I explained in my draft paper "System Identification Algorithm". The first step is calculating the initial criterion, against which the minimization will be done. For this we will need the innovation.

First, \hat{x} (the predicted state vector) is calculated by

$$\begin{aligned}\hat{x}[n+1] &= \mathbf{A} \cdot \hat{x}[n] + \mathbf{B} \cdot u[n] + \mathbf{K} \cdot \varepsilon[n] \\ \hat{y}[n] &= \mathbf{C} \cdot \hat{x}[n]\end{aligned}$$

where $\varepsilon[n]$ is the innovation which can be written out explicitly as

$$\varepsilon[n] = y[n] - \mathbf{C}[\theta] \cdot \hat{x}[n].$$

You may recognize this equation as the Kalman filter algorithm, with the exception that it assumes \mathbf{K} is a constant. \mathbf{K} , the Kalman gain, can be calculated by

$$\mathbf{K} = \mathbf{P}\mathbf{C}^T\mathbf{R}_v^{-1}$$



where \mathbf{P} is the state covariance matrix, \mathbf{C} is defined above, and \mathbf{R}_v is the measurement noise covariance matrix (all assumed constant). In our case, \mathbf{C} is constant, but \mathbf{P} and \mathbf{R}_v may not be. Thus this implementation is not a normal Kalman filter, but a simplified one. We will discuss these later in this memo.

To do the actual calculation, we can write $\hat{\mathbf{x}}$ as

$$\hat{\mathbf{x}} = \text{ltitr}\left(\mathbf{A} - \mathbf{K}\mathbf{C}, [\mathbf{K} \ \mathbf{B}], \begin{bmatrix} y \\ u \end{bmatrix}, \mathbf{x}_0\right)$$

where the n (time) and θ arguments have been suppressed for clarity and LTITR.M is a linear time-invariant time response kernel calculator that calculates the time response of the system

$$\mathbf{x}[n+1] = \mathbf{F} \cdot \mathbf{x}[n] + \mathbf{G} \cdot \mathbf{u}[n]$$

with our system represented by

$$\mathbf{F} = \mathbf{A} - \mathbf{K}\mathbf{C}, \ \mathbf{G} = [\mathbf{K} \ \mathbf{B}], \ \text{and} \ \mathbf{u} = \begin{bmatrix} y \\ u \end{bmatrix}.$$

In this case, $y[n]$ is the measurement at time n and $\hat{y}[n]$ is the predicted value of y at time n . In the case where not every translational state is measured, we have been using simulated values as the measurements. Otherwise, the algorithm as will not function, as it depends on the innovations to calculate the direction of the minimization gradient. It may be better in the future to interpolate between sensors, but this will only be useful for lower frequency modes.

Unfortunately, this method (using simulated values as measurements when a measurement is not available) can result in large errors, especially for large perturbations. When we use this method, it forces the algorithm away from the correct answer by repeatedly giving it incorrect measurement values with which to update its state equation. Depending on the value of \mathbf{K} (larger \mathbf{K} means the measurements are accurate), this can devastate the state estimation. It would be better in this case (for unmeasured states) to simply assign a value of zero to \mathbf{K} , indicating that the measurement is completely unreliable.

Now that $\hat{\mathbf{x}}$ has been calculated, we may calculate the innovations:

$$\epsilon[n] = y[n] - \mathbf{C}[\theta] \cdot \hat{\mathbf{x}}[n]$$

The least-squares minimization criterion V is then determined by

$$V = \frac{\text{trace}(\epsilon^T \cdot \epsilon)}{\text{length}(\epsilon)}$$

With the criterion for minimization defined, we proceed to the minimization loop.

Minimization loop

In the minimization loop, two major calculations occur: First, the innovation ϵ for the current value of θ is calculated just like the calculation above. Then the gradient Ψ is calculated using a finite-difference approximation to the Jacobian. This involves calculating the derivative of the predictor \hat{y} with respect to minimization parameters θ . Finally the direction of the correction is determined by solving

$$\Psi g = E$$

where E has been used for the innovation matrix. The parameters are then modified by a linear multiple of the direction g and the criterion calculated:

$$\theta = \theta_0 + \lambda g^T$$

The loop continues until a lower value of the criterion cannot be found.

The calculation of E takes place exactly as the initial calculation for the beginning value of the criterion, the only difference being that after the first loop the parameters have been modified. The calculation of Ψ , on the other hand, is more complex and has to be built up one member of θ at a time.

Changes in the algorithm required by the NTS 5 DOF model

The NTS 5 DOF model is an asymmetric 5 story building constructed by Dave and Co. that replicates many of the modal features found in the NTS data. It is a one-dimensional model that does not have rotational DOF but does incorporate shear. The challenge is to take this model, and using the data recorded at NTS, modify it so that the modal frequencies are more in line with those recorded from the actual structure.

Working with the data

In the algorithm above, the states are assumed to be the position and velocity of each node. This allows a convenient description of the problem in state-space. Our data, however, is measured in acceleration.

University of California

Originally (and somewhat naively) it was believed that we could simply filter the data with a 1 Hz highpass filter and then use a perfect integrator twice to convert to displacement, as displacement is used as the state of choice during our 5 and 50 DOF simulation trials. However, there is some noise in the signals, which for the moment we will consider white. Larry pointed out that if you integrate a noise signal with a frequency spectrum of 1, you get a noise spectrum of $\frac{1}{s}$. Do it again and now the noise spectrum is $\frac{1}{s^2}$, indicating that the noise has been significantly “reddened”, the noise at low frequencies has been increased. Depending on the noise level, it is possible to significantly distort the calculated displacement signal. Also, our modeling assumes white noise and will not operate as efficiently for reddened noise. Thus, we must either find another way to convert the acceleration data to position or change the G-N algorithm to work with acceleration, not displacement. We chose the latter.

Using acceleration instead of displacement

The first step in changing the algorithm to work with acceleration was to change the way the innovations are calculated. For this problem, modeled as a series of simple harmonic oscillators, the equations of motion are

$$\mathbf{M}\ddot{\mathbf{y}} + \mathbf{C}\dot{\mathbf{y}} + \mathbf{K}\mathbf{y} = \mathbf{u}$$

so that acceleration is related to velocity and displacement by

$$\ddot{\mathbf{y}} = \frac{\mathbf{u}}{\mathbf{M}} - \frac{\mathbf{C}_s}{\mathbf{M}}\dot{\mathbf{y}} - \frac{\mathbf{K}_s}{\mathbf{M}}\mathbf{y}$$

where \mathbf{M} , \mathbf{C}_s , and \mathbf{K}_s are the system mass, damping and stiffness matrices respectively. This means that the innovation is now represented by

$$\epsilon[n] = y[n] - \left[\frac{u[n]}{\mathbf{M}} - \frac{\mathbf{C}_s}{\mathbf{M}} \dot{y}[n] - \frac{\mathbf{K}_s}{\mathbf{M}} y[n] \right]$$

since the measurement $y[n]$ is now acceleration. The criterion is unchanged.

However, this is not the only change. Indeed, the calculation of $\hat{\mathbf{x}}$ has changed significantly due to the redefinition of the innovation. For displacement, from above:

$$\begin{aligned}\hat{\mathbf{x}}[n+1] &= \mathbf{A} \cdot \hat{\mathbf{x}}[n] + \mathbf{B} \cdot u[n] + \mathbf{K} \cdot \epsilon[n] \\ \hat{\mathbf{y}}[n] &= \mathbf{C} \cdot \hat{\mathbf{x}}[n]\end{aligned}$$

and for acceleration:

$$\hat{x}[n+1] = A \cdot \hat{x}[n] + B \cdot u[n] + K \cdot \left(y[n] - \left[\frac{u[n]}{M} - \frac{C_s}{M} x_v[n] - \frac{K_s}{M} x_d[n] \right] \right)$$

$$\hat{y}[n] = \frac{u}{M} - \frac{C_s}{M} \dot{y} - \frac{K_s}{M} y = -[M^{-1}K_s \quad M^{-1}C_s] \cdot \hat{x}[n] + M^{-1} \cdot u[n]$$

where $y[n]$ is now an acceleration measurement and $x_d[n]$ are the displacement states and $x_v[n]$ are now the velocity states. Defining

$$x_d = C_d x$$

$$x_v = C_v x$$

where C_d is the same as our old displacement C , we can write the above as

$$\hat{x}[n+1] = \left[A + \frac{KC_s C_v}{M} + \frac{KK_s C_d}{M} \right] \cdot \hat{x}[n] + \left[B - \frac{K}{M} \right] \cdot u[n] + K \cdot y[n]$$

$$\hat{y}[n] = -[M^{-1}K_s \quad M^{-1}C_s] \cdot \hat{x}[n] + M^{-1} \cdot u[n]$$

or

$$\hat{x}[n+1] = [A + KDC_s C_v + KDK_s C_d] \cdot \hat{x}[n] + [B - KD] \cdot u[n] + K \cdot y[n]$$

$$\hat{y}[n] = -[DK_s \quad DC_s] \cdot \hat{x}[n] + D \cdot u[n]$$

so that we may now write

$$\hat{x} = \text{ltitr} \left(A + KDC_s C_v + KDK_s C_d, [K \quad B - KD], \begin{bmatrix} y \\ u \end{bmatrix}, x_0 \right)$$

with $D = M^{-1}$.

Now that we have \hat{x} , we can proceed to the calculation of the G-N gradient, which is

$$\psi(n | \theta) = \frac{d(\hat{y}(n | \theta))}{d\theta} = (-[DK_s \quad DC_s] \psi_x(n | \theta) + D\psi_u) +$$

$$(-[DK'_s + D'K_s \quad DC'_s + D'C_s] \cdot \hat{x}[n] + D' \cdot u[n])$$

where ψ_x and ψ_u are the derivatives of \hat{x} and u with respect to θ and X' denotes the derivative of the matrix X with respect to θ . Since the input u is obviously not dependent on θ that term can safely be ignored. What is left now is to calculate ψ_x , the derivative of \hat{x} (Eq. 3).

$$\begin{aligned}\psi_x[n+1] = & [A + KDC_s C_v + KDK_s C_d] \cdot \psi_x[n] \dots \\ & + [A' + K'DC_s C_v + KD'C_s C_v + KDC'_s C_v + K'DK_s C_d + KD'K_s C_d + KDK'_s C_d] \cdot \hat{x}[n] \dots \\ & + [B' - K'D - KD'] \cdot u[n] + K' \cdot y[n]\end{aligned}$$

This can be calculated using LTITR:

$$\psi_x = \text{ltitr} \left[\begin{array}{c} A + KDC_s C_v + KDK_s C_d, \\ A' + K'DC_s C_v + KD'C_s C_v + KDC'_s C_v + K'DK_s C_d + KD'K_s C_d + KDK'_s C_d \dots \\ \dots K' \quad B' - K'D - KD', \end{array} \right] \begin{bmatrix} x \\ y \\ u \end{bmatrix}, \quad dx_0$$

After all these additions are made, the algorithm is almost ready to run.

The search for K

As mentioned previously, K can be calculated by

$$K = PC^T R_v^{-1}$$

where **P** is the state covariance matrix, **C** is defined above, and **R_v** is the measurement noise covariance matrix. However, this assumes **P** is constant, kind of a “half-assed” Kalman filter implementation. The Kalman filter algorithm normally does not assume **P** is constant, and calculates a new, corrected **P** at each time step:

$$P = (I - KC)P_p,$$

where **I** is the identity matrix and **P_p** is the predicted measurement for the time step. Thus **K** can be expressed as:

$$K = P_p C^T (C P_p C^T + R_v)^{-1}$$

However, we are still faced with the problem of what to use for **P_p** and **R_v**. The equation above is used at each time step in a Kalman filter algorithm, and LTITR.M does not do that. It is only an LTI kernel propagator, and assumes the **K** it is given is constant.

So really the only way to effectively implement the Kalman filter is to overhaul PEM_SPARSE.M so that it uses a full-fledged Kalman filter algorithm. This would emulate an optimal filter approach and almost certainly cause the accuracy to improve.



Sensitivity of the NTS model to changes in E

I have observed, from some of the earlier results, that a relatively large change in E as dictated by PEM_SPARSE is not reflected in the mode frequencies of the NTS structure. For example, one of my tests returned the following values:

Old E values	Calculated E values	Old mode freqs	Calc mode freqs
10.6	21.2	5.21	5.20
10.6	14.6	13.2	13.2
10.6	21.2	21.4	21.2
10.6	6.1	26.9	26.8
10.6	0.23	N/A	N/A

Table 1. Original and calculated E values along with their respective mode frequencies

It is clear that very large changes in E did not change the mode frequencies in any significant fashion. For the NTS structure, we are only varying values in the local k matrix that are proportional to EI, we are not varying the values of EA. For the other structures, we do the same thing and the sensitivity is still quite high. The only difference is that shear is included in the NTS model and not in any of the others. Perhaps this is causing the problem? It is important that we discover what is causing the problem soon, as even a perfect search algorithm will not converge to the correct answer if the structure is insensitive to the parameters that are being manipulated.

Bad data on sensor 12?

I have noticed that the data from sensor 12 (fourth floor, +x direction), while of the correct magnitude, seems to be lacking in high frequency components. In Figure 1 the PSD of sensor 12 is compared with sensors 3, 6, 9 and 15, which were oriented the same way as sensor 12. It is clear that there is not as many high frequencies in sensor 4's spectrum.

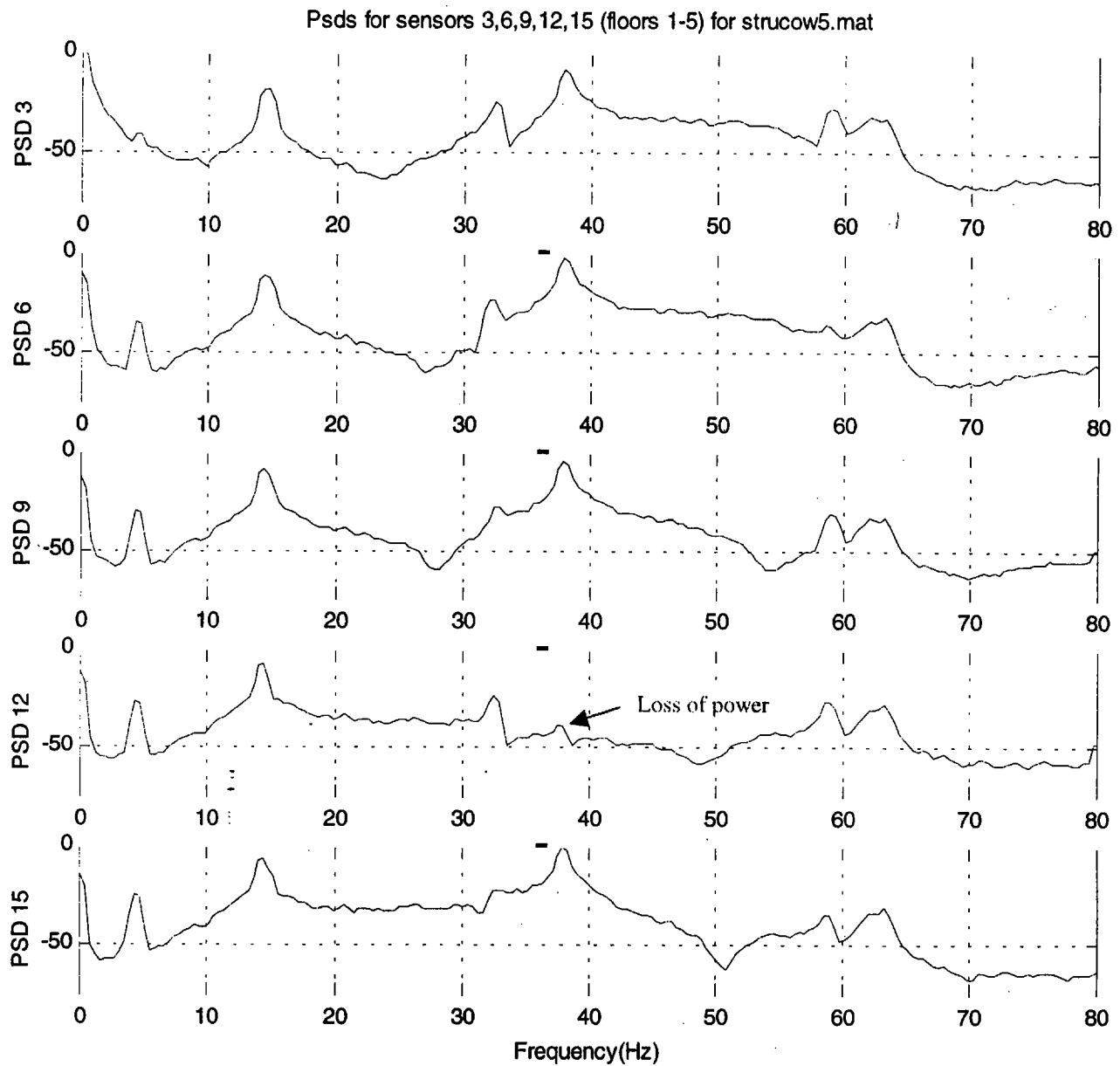


Figure 1. Power spectral densities of sensors 3, 6, 9, 12, and 15. Note the lack of power above 35 Hz for sensor 12.

Conclusions and suggestions for improvements:

The simple algorithm PEM from Matlab has been substantially changed in order to operate on the NTS structure. The acceleration is used as the input to the system so that no transformation to displacement is required. However, the algorithm can be improved by the following:

1. Implement the full Kalman filter algorithm for state estimation
 - a. Use $K = 0$ when no measurements available
2. Determine why the sensitivity of the structure with respect to EI is so low and fix it!

In addition, we may have to limit ourselves to analyzing data below about 35 Hz in order to compensate for sensor 12's inadequacies. It is not clear at this time if any of sensor 12's data is useful, we may have to try and update the model without it. As it is only one sensor, the loss should not affect the accuracy more than a few percent.

Interdepartmental letterhead

Mail Station: L-271

Ext: 33088

10/18/99

To: Greg Clark
Dave McCallen
Chad Noble
Larry Ng

From: Greg Burnett

CC:

Re: 10 and 50 DOF results with reduced measurements

Hello everyone,

I wanted to present what we have so far regarding the 10 and 50 DOF results with reduced translational measurements. We never measure the rotational states, so full measurement would imply 5 and 25 measurements respectively.

The algorithm used for these measurements is my PEM_SPARSE, which uses Matlab's PEM as a basis. It uses a damped Gauss-Newton search routine (see my earlier draft report, "system identification algo.doc") to identify the parameters. Its strong points are robustness and guaranteed convergence to a minimum (but NOT necessarily the global minimum, more on that later), but its weak points are slow convergence near the minimum and the possibility of convergence to the global minima (this possibility increases substantially as the number of DOF increases). It is also quite sensitive to the Kalman gain, and requires transforming our continuous time model to a discrete one, an operation rife with peril. However, the last one is not really its fault, as all iterative search algorithms are forced to operate in the discrete domain, so its something we will have to learn to deal with.

For this memo, the number of states that are measured is varied and the performance of the parameter identification algorithm is determined by the mean error across states for a given perturbation. Since PEM requires all states to be measured in order to function, if the states are not measured then the outputs estimated by the ORIGINAL FE model are substituted. The procedure is outlined in Figure 1.

University of California

 **Lawrence Livermore
National Laboratory**

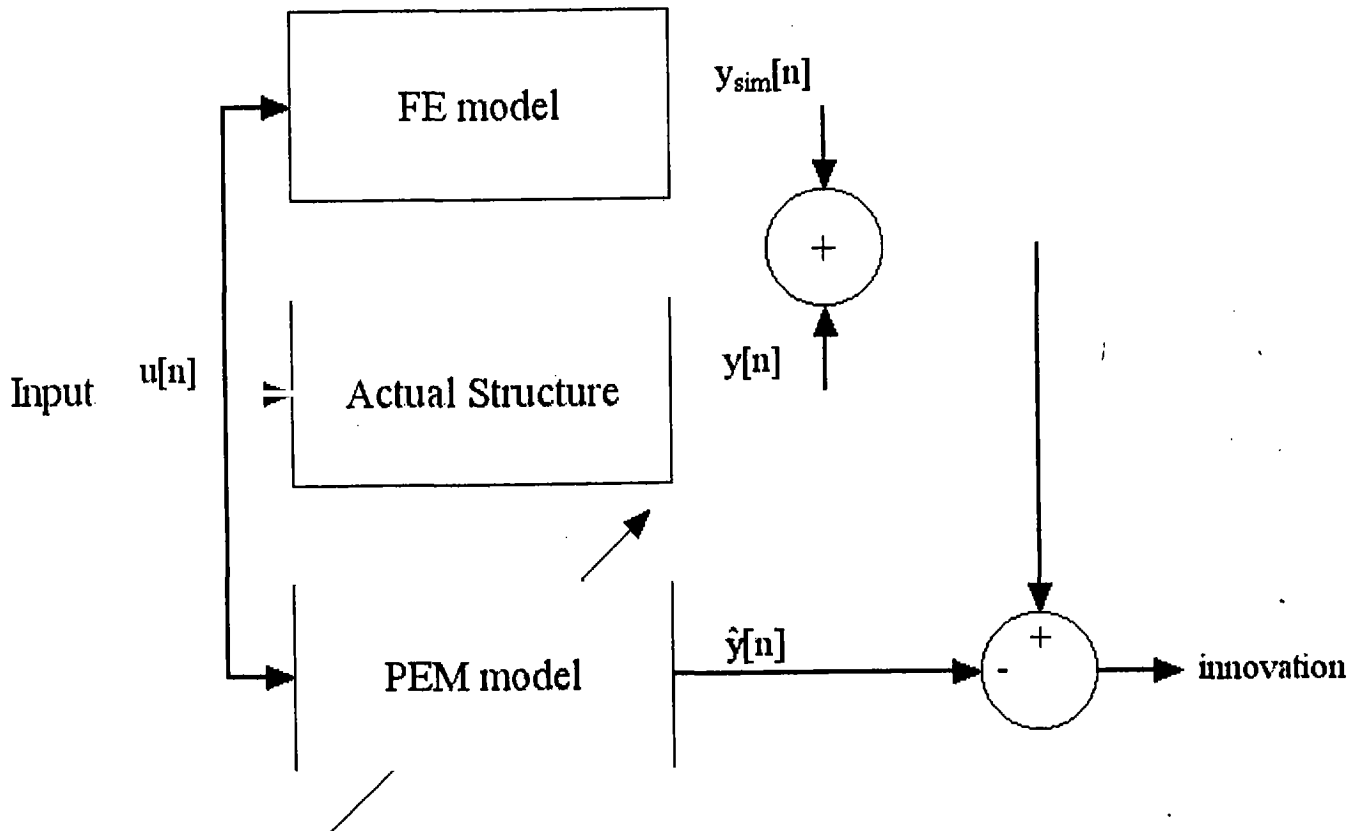


Figure 1. Flow of innovation calculation when FE model output is used in place of measured data.

We might also want to try interpolating the sensor measurements between nodes. This would only work for frequencies low enough so that the displacement between measured nodes could be interpolated with sufficient accuracy.

Unfortunately, this method (using simulated values as measurements when a measurement is not available) can result in large errors, especially for large perturbations. When we use this method, it forces the algorithm away from the correct answer by repeatedly giving it incorrect measurement values with which to update its state equation. A better way is to change the way the algorithm operates. We will address this later in the memo.

At this point it is instructive to go over in detail the difference between nodes, elements, and states. The nodes are the points in the building at which two or more elements connect. The nodes are where we place the sensors and each node may have one or more degrees of freedom. In the 10 DOF case, each

node has two degrees of freedom: an x-translation and a z-rotation. Each DOF is represented by a state, or equation of motion. It is the states (equations) in state-space that we count as our measurements. When we say node 6 was measured, what we really mean is state 6 was measured. Since there is a one-to-one correspondence at this point, it is convenient to use either “nodes” or “states”, even though they may not match up exactly. For example, node 1 of the 10 DOF model is fixed, and therefore has no state or equation associated with it. State 1 is really associated with node 2, but to avoid confusion we in essence rename our nodes starting with node 0. Thereafter node n will be associated with state n . For our simple models the nodes are considered point masses.

The elements are the physical structures bonding the nodes together. They are the ones with the physical properties such as length, width, and elasticity. In the simple 10 and 50 DOF models, each element connects two nodes. In PEM_SPARSE, the values of EI for each element are used as the variables of the system. Changing the properties of one element actually affects two nodes. Conversely, not measuring one nodes affects the calculation of two element values. The way my numbering system is set up, by not measuring node n you will affect the calculation of elements n and $n + 1$. This is important to keep in mind as we go through the following data.

10 DOF

The details of the algorithm are (these are not necessary to understand, I place them here to remind myself of what I did):

$k = [7/8, 3/4, 9/10, 1/3, 2/3] * k_0;$	% all members of k varied, some significantly
No g restraints	% no restrictions on search vector magnitude
$\text{Pars}_0/100 < \text{pars} < 2 * \text{pars}_0$	% parameters (theta vector) restrained
Kalman gain = eye(ra,r)	% Kalman gain is the identity matrix (default, not optimal)
No noise, determinant criterion	

Results:

With all states measured, the mean error across states for the parameter identification was $1.2 \times 10^{-9}\%$, essentially zero. This means the identification algorithm is dead-on, even for the large perturbations

used. As this perturbation is quite large, we can be confident that for "normal" perturbations, which should be quite small, on the order of 10%, we should be able to get close to the correct answer. The accuracy of the results will depend on the noise level.

If all the states are not measured, the results are paraphrased in Figure 2. Keep in mind that the accuracy of the G-N algorithm can vary, sometimes substantially, and these results are only two runs averaged together. Nonetheless they can give an idea of the performance versus measurement location. The cross-hatched squares denote a measured state, the open squares an unmeasured state. The results are not too surprising – since the most poorly modeled states are 4 and 5, if they are both not measured the errors are usually high: 32, 47, 43, 179% errors. However, there was one experiment where states 3, 4, and 5 were not measured and the mean error was only 4.7%. This is better than when only states 4 and 5 were not measured – 32%! This shows that for the G-N algorithm, sometimes measurements can actually hurt accuracy, depending on the model and the size of the mismatches.

Another large error was when states 1,2, and 4 were not measured (89%). When the algorithm was run again with state 3 not measured, the mean error improved to

11%. Again, measurement of state 3 can be detrimental to performance. It could be because of the large change in k between the third and fourth floors, or just due to the inconsistencies of the G-N algorithm.

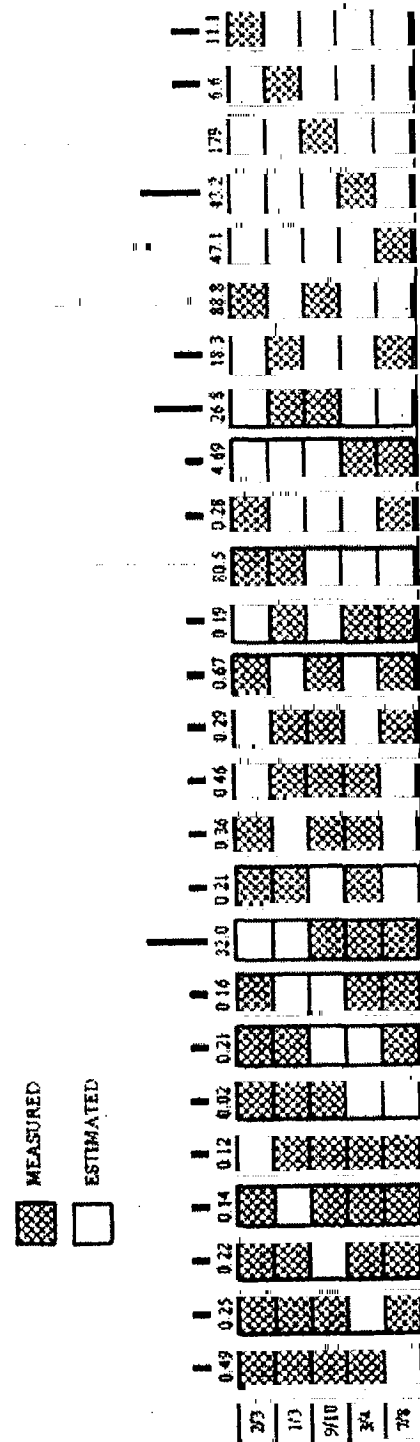


Figure 2. Mean error for various states of measurement

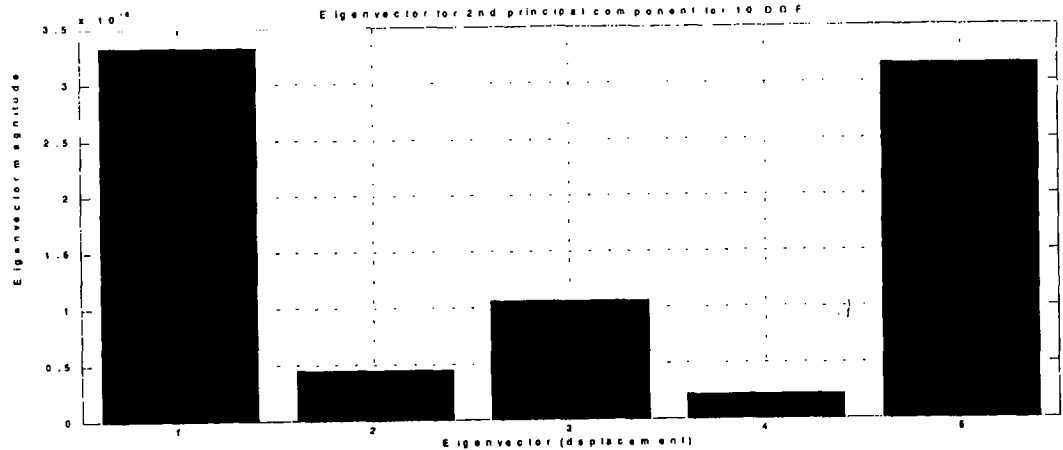
Overall, the results show that good to excellent (error rates from 0.02% to 0.67%) results for three measurements with the exception of states 4 and 5 unmeasured (32%, expected due to the large changes in k for elements 4 and 5). For two measurements there was much poorer performance (0.28, 4.69, 18.3, 26.8, 80.5, and 88.8% error) and for a single measurement the best we could do was 6.6% (state 4 measured) and the worst was 179% (state 3 measured). This seems to indicate that it is possible to model the structure relatively well with only one measurement if *it is taken in the right place*. In this case, state 4 was very poorly modeled (actual value only 1/3 of modeled value) and so a measurement there helped to converge the solution. On the other hand, state 3 was modeled the best (actual value 90% of modeled value) and so a measurement there was not always helpful – in some cases, it was actually detrimental to performance.

PCA

It has also been suggested that we may be able to look to principal component analysis to see if we can determine where we should place the sensors. Principal component analysis takes the state covariance matrix P (which we calculate in SIM_DISCRETE.M, or which we can determine from the calculated displacements vs. time) and calculates the eigenvectors (principal components) and eigenvalues. This is simply a transformation of the states to a basis where they are orthogonal and have the maximum autocorrelation and zero cross-correlation. When the transformation is complete, the principal components with negligible eigenvalues are discarded and the system can be described with the remaining principal components. If we examine the principal components with the largest eigenvalues, we can get an idea of where we should place our sensors.

For the 10 DOF system, the first two principal components are shown in Figure 3. The largest principal component indicates that states 1 and 5 may be the most important, and when we check Figure 3 we see that for states 1 and 5 measured, the error was indeed small, only 0.28%. If we take the first two principal components, it would indicate that the most important states are 1, 5, 2, and 4, which are the most poorly modeled. This would seem to indicate that PCA can tell us where to put our sensors with some accuracy, but I have had trouble with stability and repeatability of the calculation. I also will get different answers depending on the measurements available and the noise level, so this entire process must be examined in more detail before it can be implemented.

First
principal
component



Second
principal
component

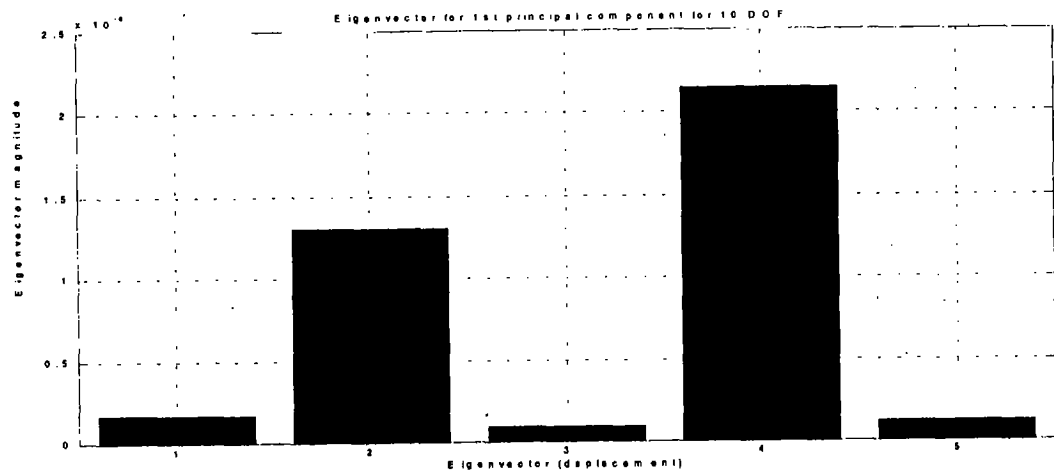


Figure 3. The first two principal components of the 10 DOF system with all translational states measured.

50 DOF

In this experiment, the columns of the previous five story building were discretized into five sections apiece, each with a translational and rotational DOF. The five floor nodes were always measured, and the location and size of the model mismatch was varied. Unlike the 10 DOF, for 50 DOF PEM_SPARSE does not converge perfectly to the actual structure in the absence of noise even with every translational node measured. The best we can usually do (for small changes in k such as 5-10%) is converge to a mean error on the 25 translational degrees of freedom of about 5%, far worse than the 10 DOF case. We will discuss possible causes of this inaccuracy later.

In the following simulations all perturbations to the stiffness elements were to reduce the stiffness (EI) constant to 90% of its nominal value. In the first set of simulations all nodes are assumed measured, and in the second only the floors (nodes 5:5:25) are measured. Keep in mind that the things that are measured are nodes, while the algorithmic parameters are the stiffness values for the elements, which are attached to other elements at the nodes. The structure is arranged so that element 1 is between nodes 1 and 2. Thus to observe node 5 is to be able to affect elements 5 and 6.

The algorithmic details are:

No g restraints, Kalman gain = 0

$\text{Pars}_0/100 < \text{pars} < 2 * \text{pars}_0$ % parameters (theta vector) restrained

No noise, trace criterion

Results (all translational nodes measured):

Ideally, the algorithm should converge to the answer with a negligible error since we are not including any noise in the process. However, the mean error varied from 0.3 % to more than 30%, without any obvious trend to the accuracy. Clearly more work is needed to improve the behavior of the algorithm when faced with large DOF systems.

Results (only 5 measurements):

The results were not too surprising, in that damage to elements connecting unmeasured nodes was not readily calculable, with nodes that are farthest from the measured nodes suffering the largest errors. The unmeasured node damage was reflected in the measured node calculated damage. The results are shown in Figures 4, 6, and 6. I have observed the following:

1. The problem of convergence is not simply an observational problem. This is illustrated by the first and final plots of Figure 6, where we see that even when observed modes are perturbed, the results are only accurate to within about 7 %. I believe this is a function of the search algorithm, as right now the update is calculated as

$$\theta = \theta_0 + \lambda g$$

where θ is the parameter vector, g is the search direction, and λ is a constant. If all but one or two of the parameters are near their optimal value, the value of λ becomes very small, making convergence to the correct θ unlikely. I would propose making λ a vector as well, thereby tailoring the update vector so that systems with many degrees of freedom can be minimized.

2. The farther away from the observed nodes, the poorer the convergence. This is expected, as the farther we move away from the measured nodes the less information we have about the actual state of the system. This cannot be avoided in the present incarnation, but the effects of the perturbations on the unmeasured nodes are noticeable on the measured nodes. For example, in the fourth plot of Figure 6, element 8 was reduced to 0.90 of its normal value. The system was unable to reduce element 8 to its correct value as it was not measured. However, to compensate, the values for elements 5-6 and 10-11 (the four elements closest to measured nodes, which the algorithm can change easily) were increased by 4.0, 3.7, 3.7, and 2.7% respectively. These are the largest errors of elements text to measured nodes in the system, and reflect the perturbation of the element located between them. This phenomenon occurred each time significant perturbations were made to the structure between measured nodes. It could be quite useful as a diagnostic tool and for generally locating model mismatches in large models.
3. The convergence is still not good for many perturbations. In Figure 4, the first and second plot, it is clear that the convergence is only good near nodes 10 and 15. Near the others the error is still large. This would seem to indicate that the best convergence occurs in the middle of the structure. It also demonstrates that if many elements are perturbed or poorly modeled, the entire structure will not converge readily.

Conclusions

The Gauss-Newton algorithm may not be the best one for our purposes. It works well for a small number of DOF, but is not be sensitive enough to correct localized disturbances in large DOF structures. In addition, it is too sensitive to small changes in the Kalman gain, exhibiting widely varying

characteristics depending on the gain chosen. It also has many variables – among them the Kalman gain, the limits on the search direction magnitude and parameter values, a robustification parameter, and the numerical differentiation step size. I do have some ideas for improving its performance (see *The 5 DOF NTS model using the damped Gauss-Newton algorithm* memo of 10/18/99), but I am beginning to feel that it may be a good idea to spend some time looking at other methods which easier to work with. I will spend the next few days reviewing the different techniques available to us, and my next report will detail what I have found.

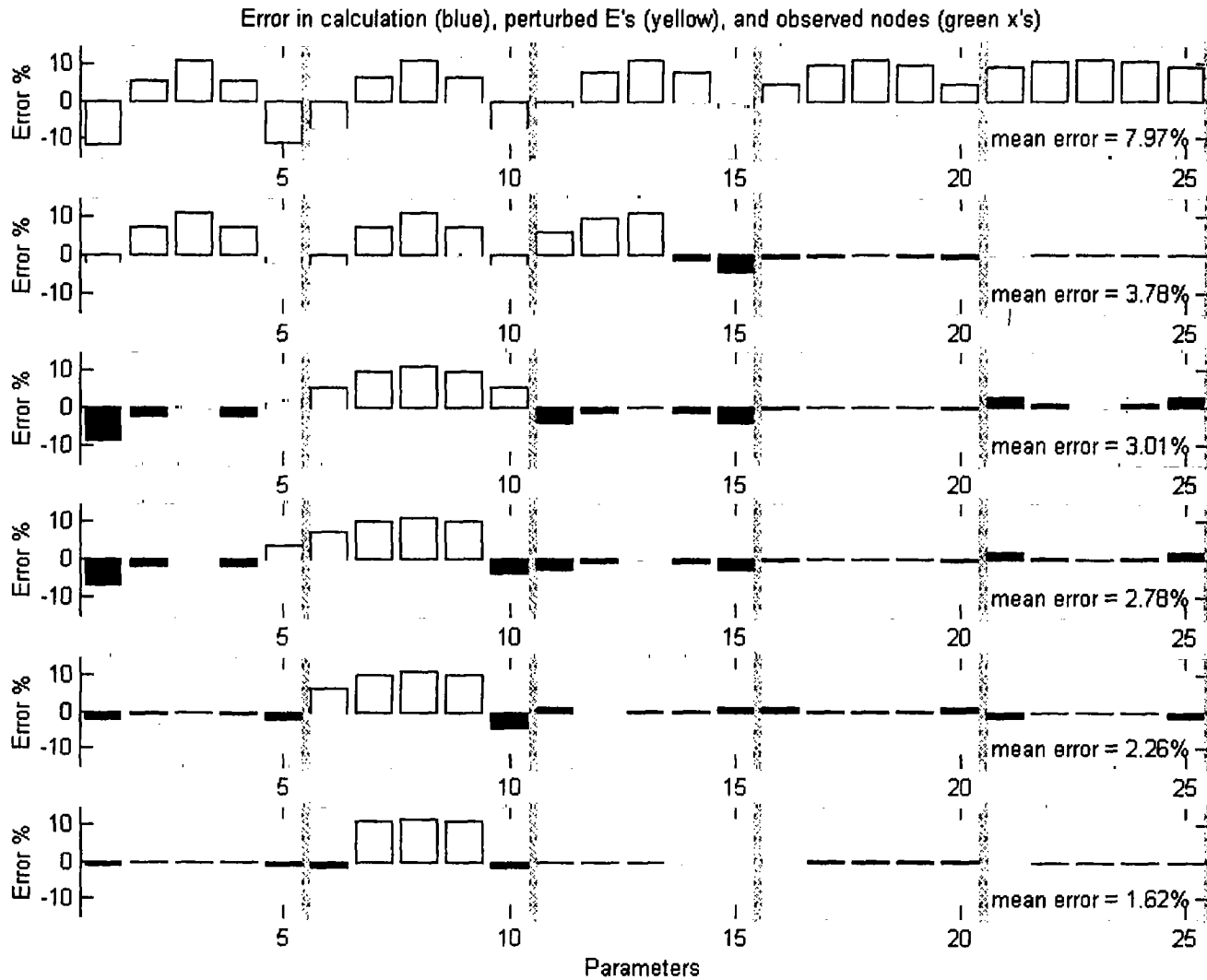


Figure 4. Calculation errors for the 50 DOF problem with 5 observations. The errors are shown in the bar graph with the yellow bars denoting elements that were perturbed to 90% of their nominal value and blue bars denoting unperturbed elements. The green lines indicate where the measurements took place.

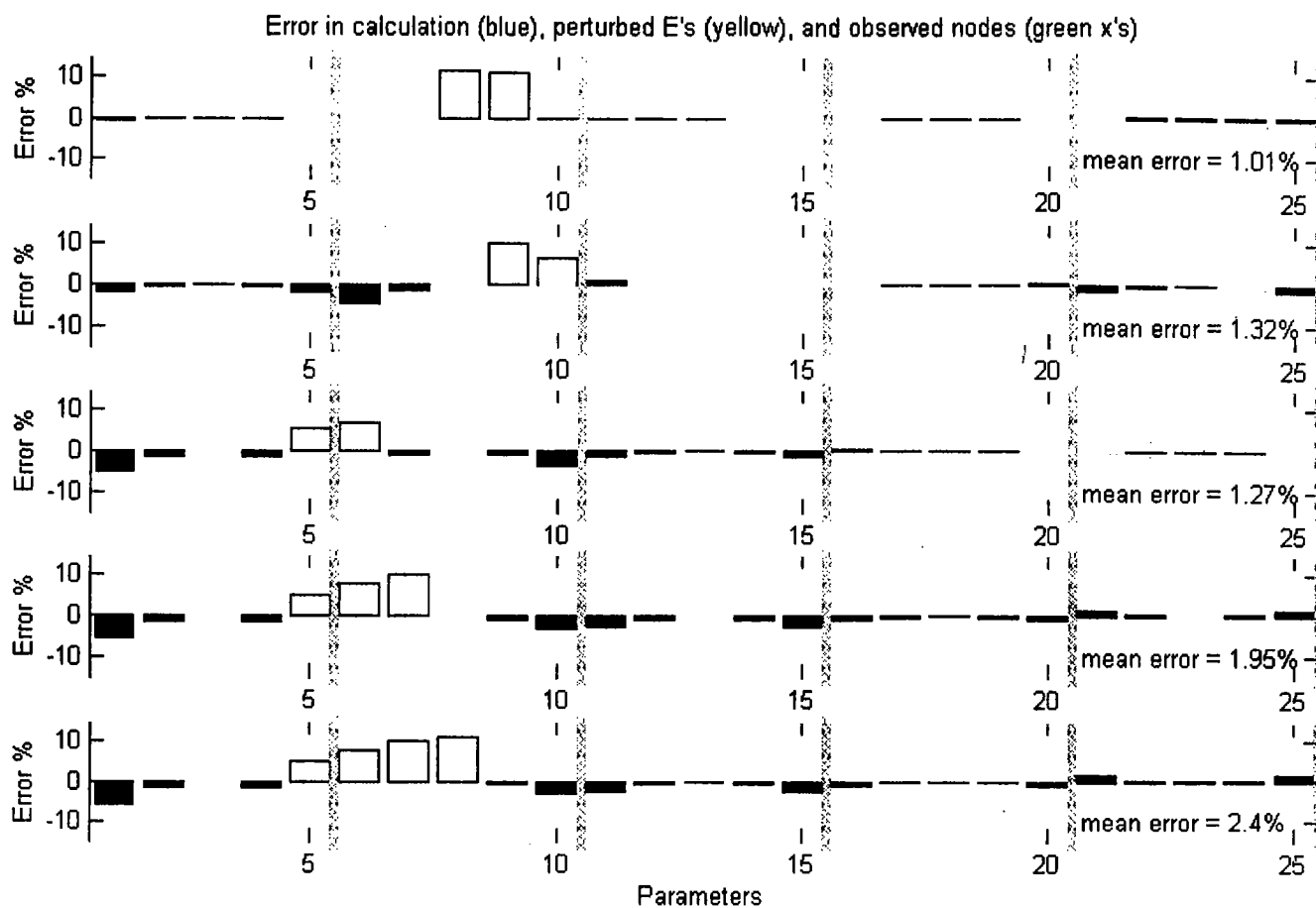


Figure 5. Calculation errors for the 50 DOF problem with 5 observations. The errors are shown in the bar graph with the yellow bars denoting elements that were perturbed to 90% of their nominal value and blue bars denoting unperturbed elements. The green lines indicate where the measurements took place.

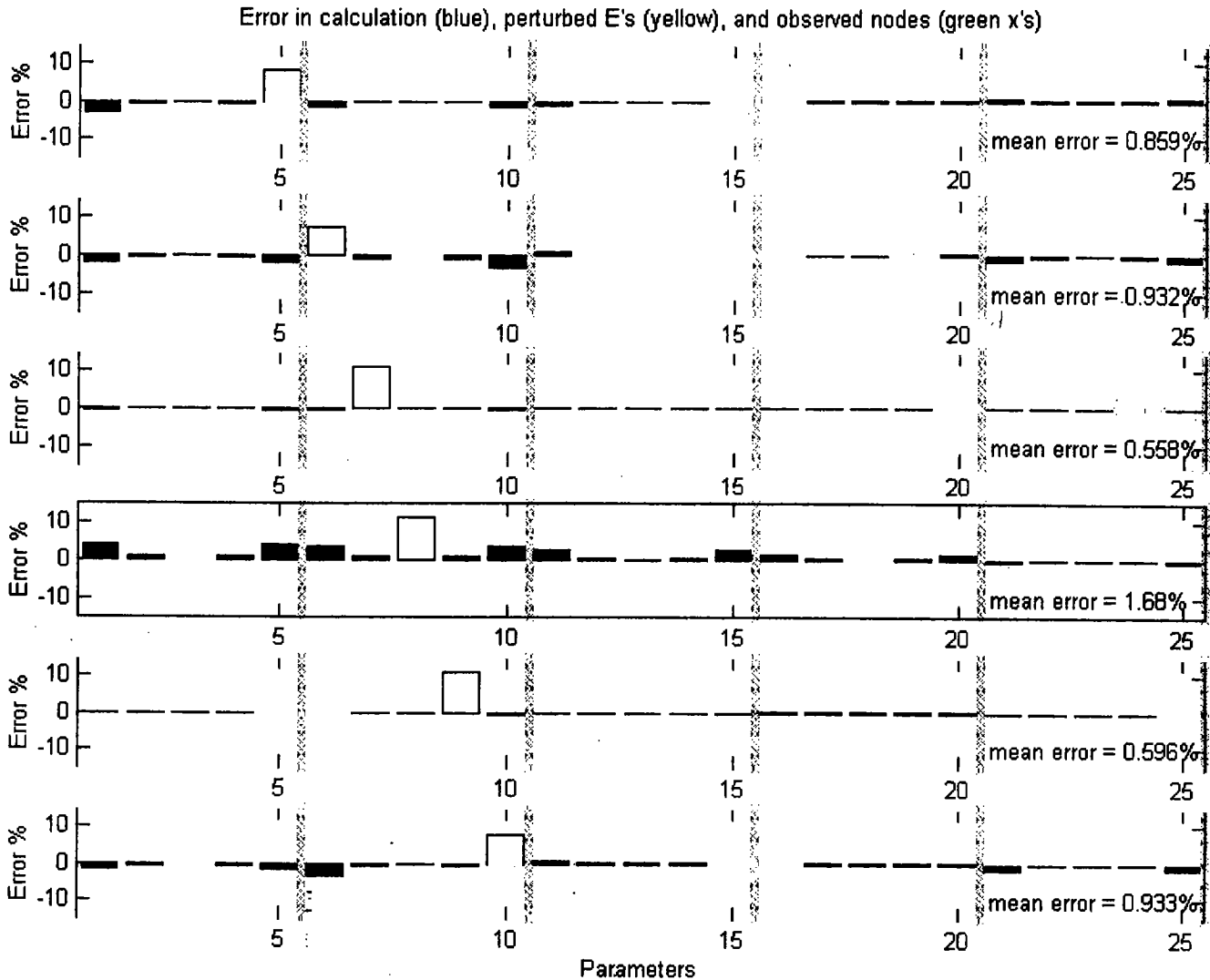


Figure 6. Calculation errors for the 50 DOF problem with 5 observations. The errors are shown in the bar graph with the yellow bars denoting elements that were perturbed to 90% of their nominal value and blue bars denoting unperturbed elements. The green lines indicate where the measurements took place.

Interdepartmental letterhead
 Mail Station: L-290
 Ext: 33088

12/22/99

To: Larry Ng
 Dave Harris
 Greg Clark
 Dave McCallen

From: Greg Burnett

CC:

Re: Augmented state vector continuous-discrete extended Kalman filter system identification approach

Hello everyone,

This is to present the approach Larry has suggested that we use for system identification. It augments the state vector (normally just the displacements and velocities of the system) with the stiffness parameters. This forms a nonlinear system, which is then approximated using an extended Kalman filter. The system is described in continuous time, and the measurements are assumed to take place discretely. Most of this information was taken from Gelb [1].

A nonlinear system can be represented as

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), t) + \mathbf{w}(t) \quad (1)$$

$$\mathbf{y}[k] = \mathbf{h}(\mathbf{x}(k), k) + \mathbf{v}[k] \quad k = 0, 1, 2, \dots \quad (2)$$

with $\mathbf{Q}(t)$ and $\mathbf{R}[k]$ as the covariance matrices of $\mathbf{w}(t)$ and $\mathbf{v}[k]$. We want to calculate the minimum variance estimate of $\mathbf{x}(t)$ as a function of time and measurement data. In this system, the states propagate in continuous time, and measurements occur at discrete times. Therefore just after a measurement has been taken, the states propagate according to the equations above until the next measurement occurs and the states are adjusted accordingly. In order to represent this process, we use the following nomenclature:

$\hat{\mathbf{x}}$ = measurement vector

$\mathbf{x}[k_+]$ = state just after measurement

$\mathbf{x}[k_-]$ = state just before measurement

This will also be used for P , the error covariance matrix, which we are trying to minimize.

After each measurement, Equations 1 and 2 are linearized about the measurement and the states propagated to the next measurement time. A first order approximation is made for (1), and a second order approximation is made for the state covariance matrix P . This leads to

$$\begin{aligned}\dot{\hat{\mathbf{x}}}(t) &\approx \mathbf{f}(\hat{\mathbf{x}}, t) \\ \dot{\mathbf{P}}(t) &\approx \mathbf{F}(\hat{\mathbf{x}}, t)\mathbf{P}(t) + \mathbf{P}(t)\mathbf{F}^T(\hat{\mathbf{x}}, t) + \mathbf{Q}(t)\end{aligned}\quad t_{k-1} \leq t < t_k \quad \begin{matrix} (3) \\ (4) \end{matrix}$$

where

$$\mathbf{F}_{ij} = \left. \frac{\partial f_i(\mathbf{x}, t)}{\partial x_j} \right|_{\mathbf{x}(t) = \hat{\mathbf{x}}(t)}$$

So before each measurement point, $\mathbf{P}[k_-]$ and $\hat{\mathbf{x}}[k_-]$ are calculated according to 3 and 4 above. To update the system following a measurement at time k , the following steps are taken:

- 1) Calculate the Kalman gain: $\mathbf{K}[k] = \mathbf{P}[k_-] \mathbf{H}_k^T (\hat{\mathbf{x}}[k_-]) \cdot [\mathbf{H}_k (\hat{\mathbf{x}}[k_-]) \mathbf{P}[k_-] \mathbf{H}_k^T (\hat{\mathbf{x}}[k_-]) + \mathbf{R}[k]]^{-1}$
- 2) Update the state equation: $\hat{\mathbf{x}}[k_+] = \hat{\mathbf{x}}[k_-] + \mathbf{K}[k] \cdot [y[k] - h(\hat{\mathbf{x}}[k_-], k)]$
- 3) Update the state covariance: $\mathbf{P}[k_+] = [\mathbf{I} - \mathbf{K}[k] \mathbf{H}_k (\hat{\mathbf{x}}[k_-])] \cdot \mathbf{P}[k_-]$

where

$$\mathbf{H}_{ij} = \left. \frac{\partial h_i(\mathbf{x}, t)}{\partial x_j} \right|_{\mathbf{x}(t) = \hat{\mathbf{x}}(t)}$$

The states are now propagated according to Equations 3 and 4 to the next measurement point. It is clear that the measurement points are not required to be regular nor even to occur at all, but given relatively noise-free measurements, the more that can be taken, the better the state estimation.

For a 1-D problem the states are defined as

$$\mathbf{x} = [x \quad \dot{x} \quad k]^T$$

with x representing the displacement of the SHO and k the stiffness. The equations describing their evolution are

$$\begin{aligned}\dot{x}_1 &= x_2 \\ \dot{x}_2 &= -\frac{x_1 x_3}{m} - \frac{b}{m} x_2 + \frac{u}{m} \\ \dot{x}_3 &= 0\end{aligned}\quad (5)$$

so that

$$\mathbf{F} = \begin{bmatrix} 0 & 1 & 1 \\ -x_3/m & -b/m & -x_1/m \\ 0 & 0 & 0 \end{bmatrix}$$

and

$$\mathbf{H} = [-x_3/m \quad -b/m \quad -x_1/m]$$

The only assumptions made in this algorithm are that the system and measurement noises are uncorrelated, the initial state is a Gaussian with the given initial state as the mean and P_0 as the variance. I have implemented this algorithm in Matlab (EKFSHO.M), with only one DOF and using ODE15S.M to do the propagation between measurements. For the noise matrices, I used

$$\mathbf{P} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 100 \end{bmatrix}$$

$$\mathbf{Q} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$R = 0.01$$

for the initial error, state noise, and measurement noise covariance matrices. The choice of initial \mathbf{P} reflects the relatively large uncertainty in the third state, the stiffness parameter. A large value of $P(3,3)$ tells the algorithm that any error that may occur is most likely due to the stiffness parameter, which is state 3. The choice of \mathbf{Q} assumes that I am quite certain of the development of the displacement, velocity, and k vectors. This is equivalent to saying that I am perfectly certain that the system evolves as Equations 3 and 4 specify, and there is no additive white noise to the system evolution. This is a good estimate for the present example, but may have to be changed for systems where the true model is not known and whose propagation model is not complete. The larger the value of \mathbf{Q} , the longer the algorithm takes to converge on the actual state (if it is too large the algo will not converge). As for R , a value of 0.01 says that I can measure the acceleration without bias to within about 0.1 m/s^2 , a reasonable amount.

The results of the algorithm are shown in Figures 1-4. Figures 1 and 2 demonstrate the convergence for a sine wave input of 1 Hz, and Figures 3 and 4 show the behavior for a white noise input. The simulated states and measurements are shown in blue, the estimated ones in red. At this point the algorithm is relatively slow, (about 2 minutes for a thousand data points and 1 DOF, unoptimized) but does manage

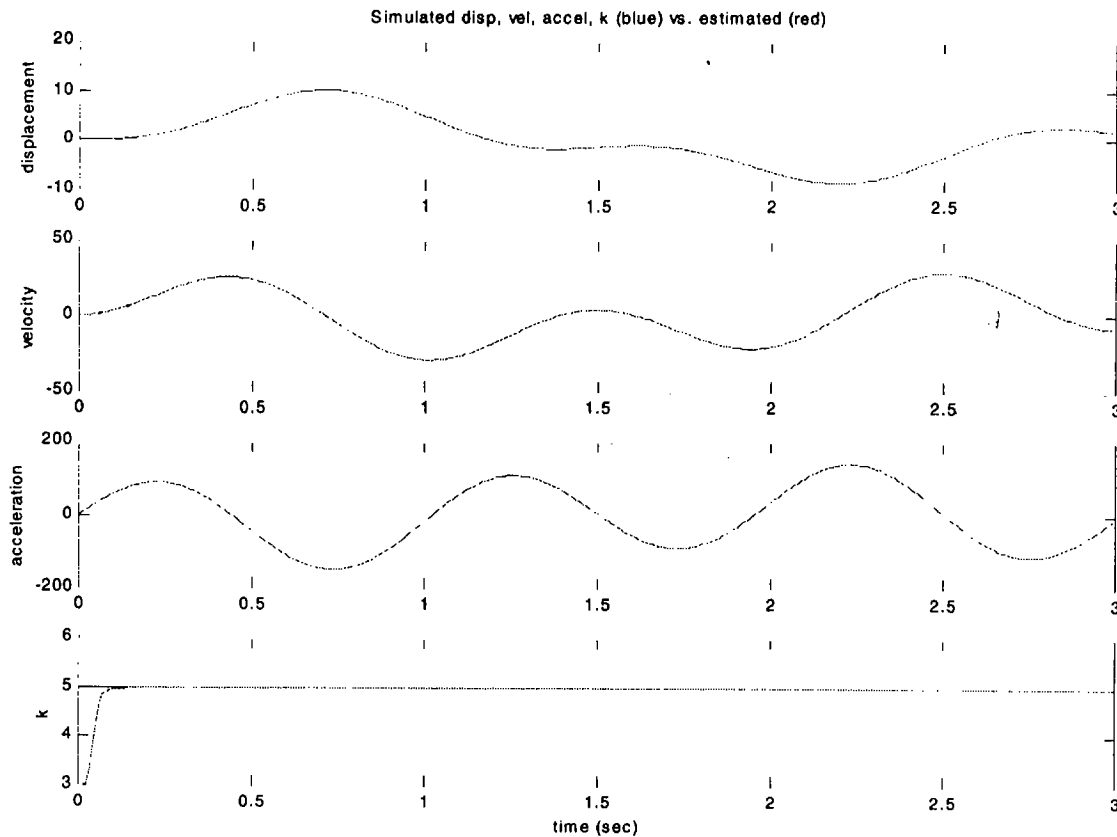


Figure 1. Demonstration of convergence of the state vector (top two, fourth graphs) and the acceleration estimate (third graph) for a sinusoidal input of 1 Hz with max amplitude of 100. The actual states and accelerations are shown in blue, the estimates in red. Note the convergence occurs very quickly, in this case after about 100 measurements. File: EKFSHO.M

to track the displacement, velocity, and k states almost perfectly within a few hundred samples with almost any initial guess for k . It will converge for almost any amplitude and frequency of sine wave input, although it converges more quickly for larger amplitudes, as $H(3)$ depends directly on the displacement, and that is the major contributor to $K(3)$.

One caveat is that the system is not constrained to converge to the correct answer, and indeed cannot if the noise terms Q are too large. The way to check if the system has settled on the correct answer is to look at K vs. time. If the members of K go to zero (or really to the value of R), the system is settling on the correct answer. If not, K is simply being used as a crutch to match the output with incorrect states.

Another problem is how to treat the input u . For the simulation, the input is discrete, like the measurements, but in equations (5) above, it is clear that the derivative of x_2 depends on u . Thus, when the ode solver is called, it should be supplied with a continuous u across the entire interval of interest. We do not have that, nor is there a place in the ode algorithms for it, as they assume undamped systems.

So, for now I am interpolating u by a factor of 100, then inside ODE15S I am using the interpolated values. This seems to work quite well, as the frequencies of interest are below about 10 Hz, so u should appear to be continuous to ODE15S.

This algorithm has a lot of promise as it estimates the state vector directly, and operates in the continuous domain (although the measurements are considered to be discrete, a perfect match for our situation). Thus no continuous to discrete transformation is necessary. The algorithm is also recursive, keeping memory requirements down. Being recursive, given "good" data (a high enough signal to noise ratio) the estimate gets better as data is processed, and the estimation can be terminated after reaching a certain criterion. There is no gradient to calculate, and no local minima to get caught in. As long as the model and the measurements are relatively good and the noise estimates on the right order, this algo should converge. However, I am not sure how it will work with systems that are not well observed.

The next step will be generalizing the algorithm to work with MIMO (multiple input multiple output) systems. I will begin that when I return from vacation.



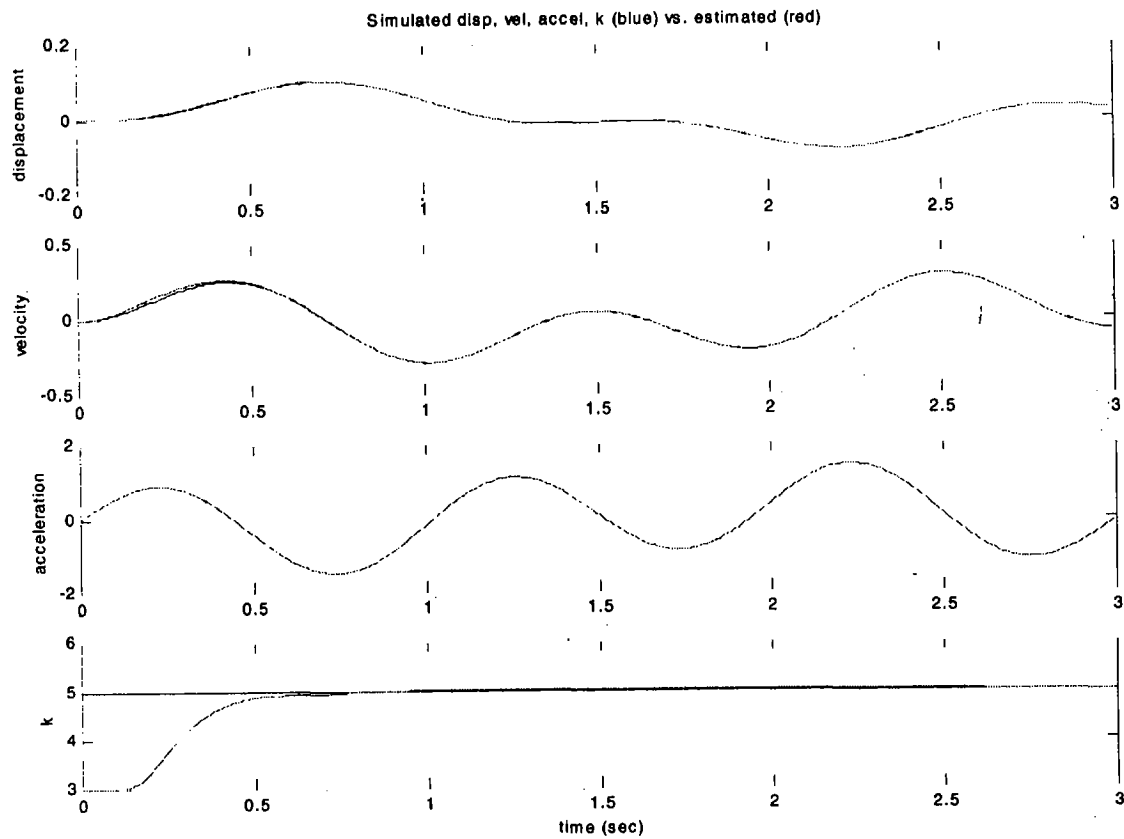


Figure 2. Demonstration of convergence of the state vector (top two, fourth graphs) and the acceleration estimate (third graph) for a sinusoidal input with max amplitude of 1. The actual states and accelerations are shown in blue, the estimates in red. Note the convergence occurs more slowly for this lower amplitude as compared to Figure 1.

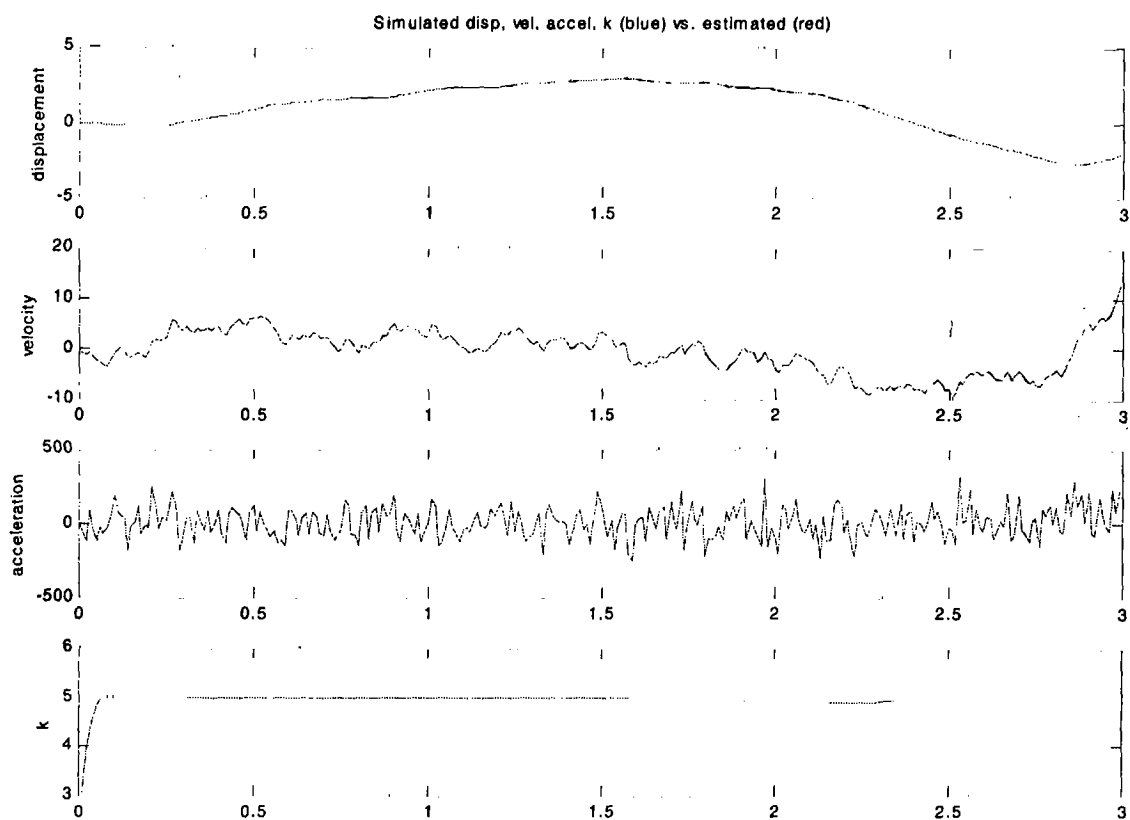


Figure 3. Convergence for a white noise input, max displacement of 100. The time of convergence is about the same as the sinusoidal input with max amplitude of 100.

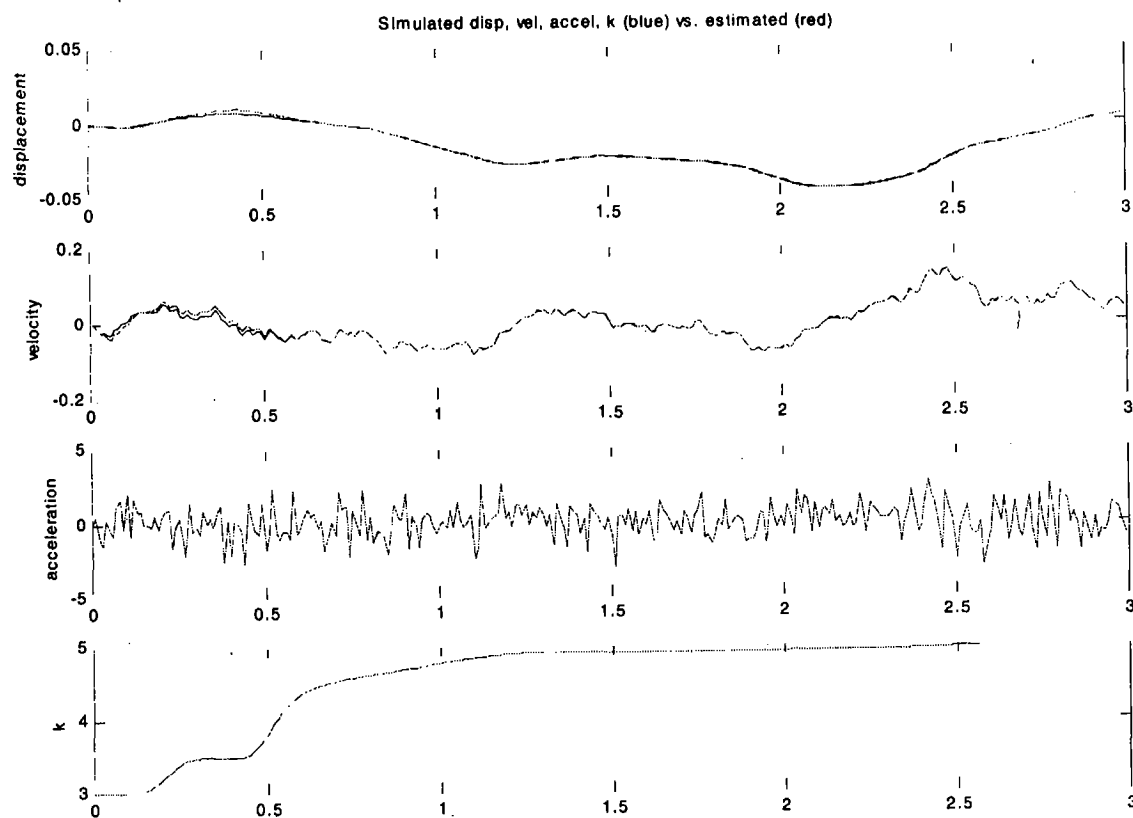


Figure 4. Convergence for a white noise input, max displacement of 1. The time of convergence is slower than the sinusoidal input with max amplitude of 1.

Reference:

- [1] Gelb, A. editor (1999). "Applied Optimal Estimation" M.I.T. Press, Cambridge, MA

Interdepartmental letterhead
Mail Station: L-290
Ext: 33088

01/04/00

To: Larry Ng
Dave Harris
Greg Clark

From: Greg Burnett

CC: Dave McCallen

Re: The dependence of convergence time for the EKF on the various noise parameters

I have studied the extended Kalman filter algorithm EKFSHO.M (see memo of 12/22/99) which operates on a single DOF system in detail. I have noticed that it can be quite sensitive to some of the given noise parameters, which include:

R = measurement noise covariance. This is a measure of how good the acceleration measurement is. It is assumed to be of zero mean as we can HP filter any bias out. Our nominal value for R will be about 1% of the maximum acceleration amplitude, a reasonable accuracy estimation.

Q = state propagation noise covariance. This is a measure of how well the states propagate according to the propagation equations given (the model). It, like all the other noise parameters, assumes the disturbances to the propagation are white with the covariances of each state on the diagonal. However, poor or undermodeled systems will likely have correlated propagation errors, which may cause convergence problems. This will have to be investigated and allowances made if necessary. In our single DOF system, of course, the model is perfect, so all of these values should be quite low. We will take as a starting point values that are about 1-10% of the maximum amplitude of the states.

P = error covariance. This is a measure of the size of the error for each state. We supply only the initial estimate, and the updated estimate is calculated in the algorithm. Our estimate should reflect our confidence in the state calculation. Our initial estimate is low (around 1-10%) for the states of displacement and velocity, but high (on the order of the estimated k) for the stiffness state as it is our unknown.

University of California

 **Lawrence Livermore
National Laboratory**

Setup

A sine wave at 1 Hz with an amplitude of 1 Newton is used as the input to a SHO with $m = 1$, $b = 0.3$, and $k = 5$. The estimate of k was 3, an initial error of 40%. The sampling frequency was 100 Hz. This system is underdamped and has a natural frequency of 0.36 Hz. Earlier tests have shown that the convergence time is inversely related to the amplitude and frequency. That is, all other things being equal, the higher the amplitude and frequency, the shorter the convergence time.

For each experiment, a standard set of noise parameters (see Table 1) was selected and a single noise parameter was changed and its effect on the convergence time was noted. In each Figure, a single noise parameter is plotted vs. the convergence time. The convergence time was defined as the time required for the k estimate (state 3) to converge to within 2% of the actual value. If there were oscillations, the time is judged to be the last time that the plot crossed the $\pm 2\%$ lines before it stayed completely within the lines. Note that the plots are logarithmic on the x axis and that the y scales (time) are in seconds and are frequently quite different. I'll address them each in turn.

parameter	R	Q ₁₁	Q ₂₂	Q ₃₃	P ₁₁	P ₂₂	P ₃₃
default	0.01	0.001	0.01	0.1	0.01	0.01	10

Table 1. Default values for the noise covariance parameters used in the experiments.

Analysis

Figure 1: R. As expected, the convergence time is intimately related to the amount of noise in the measurements. For very little noise, the convergence time is on the order of 2.4 seconds or 240 samples. For a covariance of 0.01 (which implies an accuracy of $\pm\sqrt{0.01} = \pm 0.1 \text{ m/s}^2 = \pm 0.01 \text{ g's}$), the convergence time is essentially unchanged. For a covariance of 0.1 (accurate to 0.32 m/s^2 , or about 0.03 g's) the convergence time jumps to 4.6 seconds. For a covariance of 1 (about 0.1 g's) the convergence increases to 15 seconds. It is clear, then, that accelerometer performance will be crucial in keeping convergence times down. I believe we can expect errors on the order of about 0.01 g's , but I'll have to talk to the MEs to confirm this.

Figure 2: Q₁₁. This is the estimated noise of the propagation of state 1, the displacement.

Since our model is perfect, we expect the convergence time to increase with noise level and this

University of California

is what we observe. The convergence time increases dramatically after a covariance level of about 0.0001, or about 0.01 meters. As the maximum displacement of the SHO was on the order of 0.1 meters in this experiment, as long as our propagation noise is below about 10% (covariance of 1/100) of the maximum displacement, the convergence will be rapid.

Figure3: Q_{22} . Here we would expect similar results to those seen for Q_{11} , as this state is the velocity of the system which should be described perfectly by our model. We see the same behavior, but the plateau at short times is not as pronounced. Here, for noise levels below about 0.001 or 0.03 m/s, convergence is excellent. As the maximum velocity is about 0.3 m/s, this indicates that again convergence is rapid if the noise level is below about 10% (covariance of 1/100) of the maximum of the state.

Figure 4: Q_{33} . This plot is a little different, as this state is the stiffness parameter. Instead of rising with noise level, it drops, from 2.9 seconds at a covariance of 0.0001 to 1.1 seconds at a covariance of 1000-10000. Although we are quite sure that the stiffness of the SHO does not change, and therefore this noise propagation should be very small, the algorithm works faster if it is very large. Perhaps the large noise estimate allows the algorithm to change k more rapidly than would be possible with a small P_{33} . Indeed, if we examine the P propagation, we see that

$$\dot{P}(t) \approx F(\hat{x}, t)P(t) + P(t)F^T(\hat{x}, t) + Q(t)$$

which would allow our P_{33} to be larger with each time step. As K is directly proportional to P

$$K[k] = P[k_-]H_k^T(\hat{x}[k_-]) \cdot [H_k(\hat{x}[k_-])P[k_-]H_k^T(\hat{x}[k_-]) + R[k]]^{-1},$$

(the quantity inverted in this case is a scalar) and the estimate of the state is proportional to K ,

$$\hat{x}[k_+] = \hat{x}[k_-] + K[k] \cdot [y[k] - h(\hat{x}[k_-], k)]$$

then the estimate of k will converge more rapidly if P_{33} is large. As the convergence plateaus after about 1000, an uncertainty 10 times larger than the estimate (covariance of about 100) is probably enough. Much larger than that and the solution begins to oscillate and does not smoothly converge to the correct answer.

Figures 5 and 6: P_{11} and P_{22} . These two plots are analyzed together because their behavior is quite similar. As this is just the initial estimate of the error covariance, it shouldn't have a large influence on the convergence time as the initial displacement and velocity errors are quite small (on the order of 10^{-3} for the displacement and 10^{-2} for the velocity for $k = 3$). Indeed, for these two parameters the convergence time is quite insensitive to the initial estimate. For P_{11} ,

there is almost no change as the estimate is varied from 10^{-4} to 10. For P_{22} , there is a slight change from 2.5 seconds at 10^{-4} to 4 seconds at 10, but this is insignificant compared to the changes in convergence time for members of Q . We conclude that the initial values for these equations are not critical, and will use about $(0.1)^2$ times the maximum displacement and velocity expected for the system.

Figure 7: P_{33} . This parameter identifies the relative confidence we have in the estimate of k . Setting this to a large relative value indicates we are unsure of our estimate and allows the algorithm to change the estimate more rapidly. It is clear that there is a substantial change in convergence time when P_{33} is varied from 0.1 to 10^4 . The major change occurs below about 100, though, which is on the order of our estimate. Therefore we conclude that initial values for P_{33} should be about $(1-10)^2$ times the initial k estimate.

Conclusion

For this system the following values have shown to result in minimum convergence times when tested individually:

Parameter	R	Q_{11}	Q_{22}	Q_{33}	P_{11}	P_{22}	P_{33}
Value	0.01	0.0001	0.001	100	0.0001	0.001	1000
Relative	± 0.01 g's	(10% of displacement) ²	(10% of velocity) ²	$(1-10 \times k \text{ estimate})^2$	(10% of displacement) ²	(10% of velocity) ²	$(1-10 \times k \text{ estimate})^2$

Table 2. The values chosen for the noise parameters and their relationship to the expected values.

The convergence of this system is only 0.25 seconds, compared to 2.5 seconds with our default values, a decrease by a factor of 10. So it is clear that the values used for the noise parameters can affect the performance of this system significantly.

So what does this mean for us? How is this better than the Gauss-Newton approach that I detailed in an earlier memo? There are many advantages to this algorithm:

1. It models the problem as a continuous system with discrete measurements, negating the need to transform the continuous system to a discrete one.
2. It is a recursive algorithm, so the data is processed sequentially, reducing the memory requirements. This is also likely to be how the algorithm is implemented in an actual application.



3. The measurements modeled are the accelerations, not the displacements. Therefore no conversion of data is needed.
4. The noise characteristics required can be used by the algorithm to adjust the model to fit a variety of situations, even ones where the model may be inadequate.
5. Even with very large initial estimate errors (on the order of 10000%), the algorithm can converge on the correct answer.
6. Measurements at every node are not necessary, the algorithm will just continue to propagate the model according to its properties.

Disadvantages:

1. The continuous propagation of the system requires the use of ODE solvers, which can be slow.
2. Without sufficient excitation (low amplitude and frequency) of the structure, the algorithm may never converge onto the correct answer. For example, with a sine input of amplitude 100 an initial estimate of $k = 1000$ converged to the correct answer of $k = 5$ in 1 second. With the same sine input with amplitude 1, the algorithm never converged. This is an extreme example, but this type of behavior is important to know about.

Unknowns:

1. How rapid the convergence will be for large DOF systems.
2. How rapid the convergence will be for sparsely observed systems.
3. What the "zone of influence" around each measured node will be. That is, how far away from each node will we be able to identify the stiffness parameters.
4. How fast or slow the algorithm will be for large DOF systems.
5. Whether or not simplified models of complex systems (i.e. 5 DOF NTS model) will be accurate enough to tell us anything.

I will now move to the 5 DOF system, and will perform a similar test on it as well as examine the effects of observability and controllability.

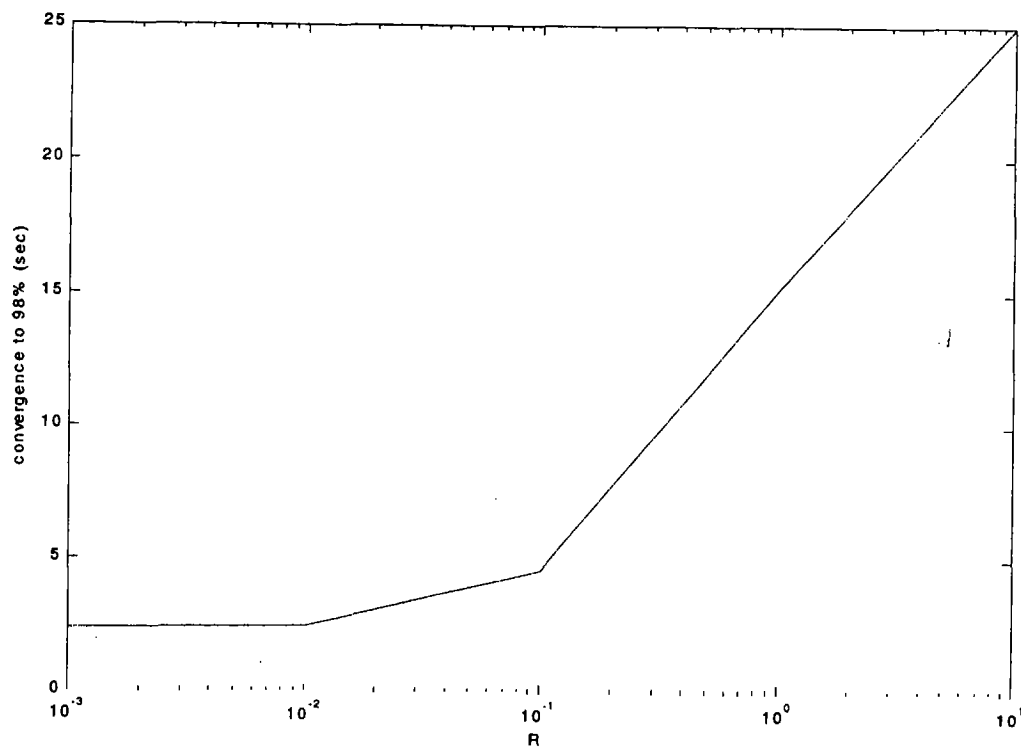


Figure 1. Plot of R vs. convergence time to within 2% of the actual k value.

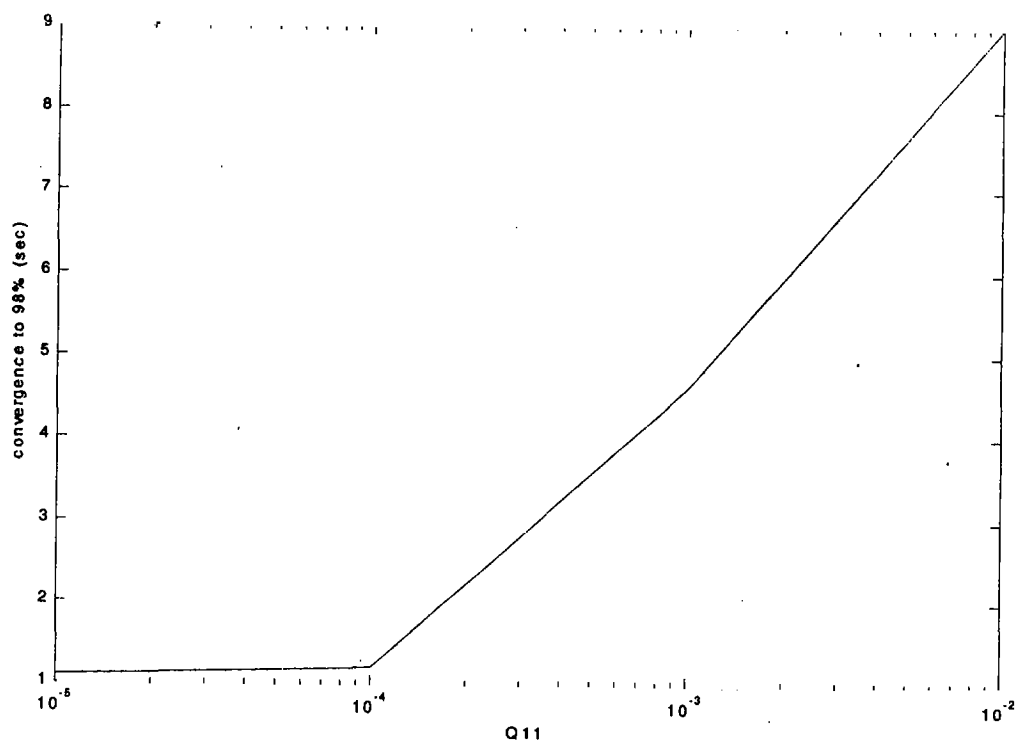


Figure 2. Plot of Q_{11} vs. convergence time to within 2% of the actual k value.

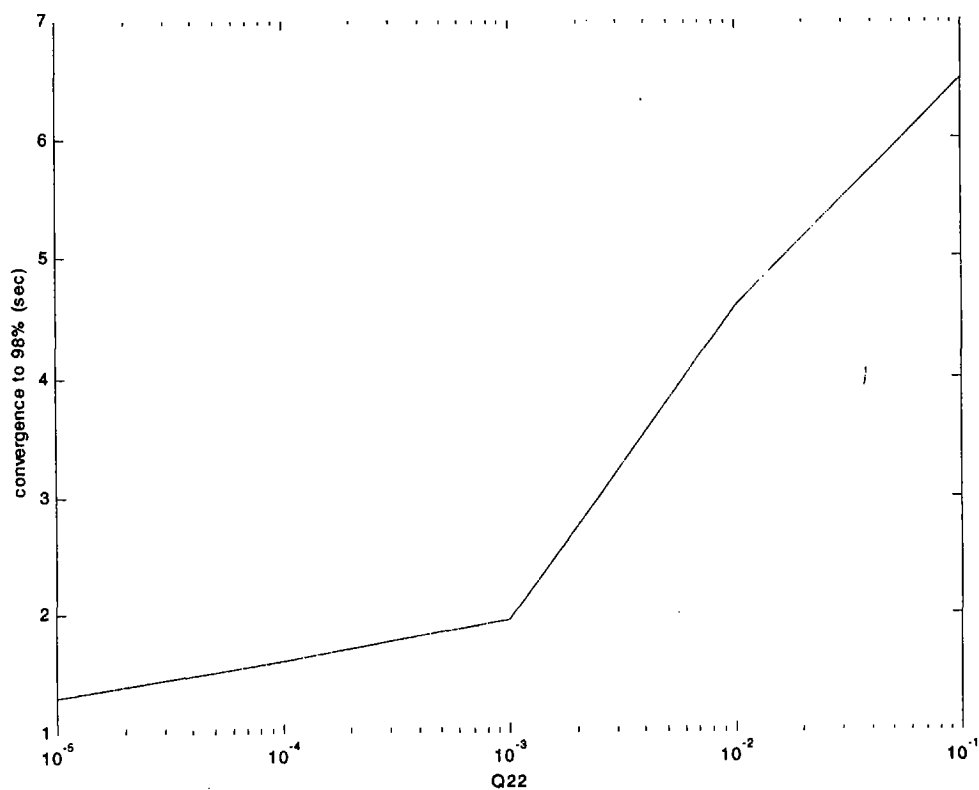


Figure 3. Plot of Q_{22} vs. convergence time to within 2% of the actual k value.

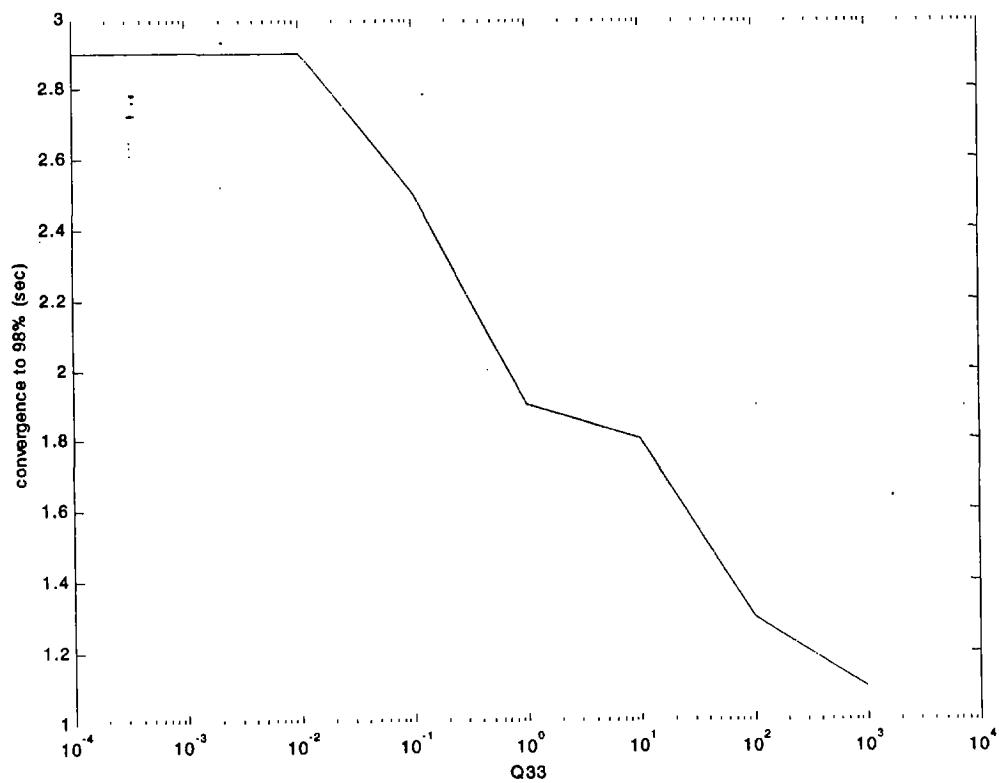


Figure 4. Plot of Q_{33} vs. convergence time to within 2% of the actual k value.



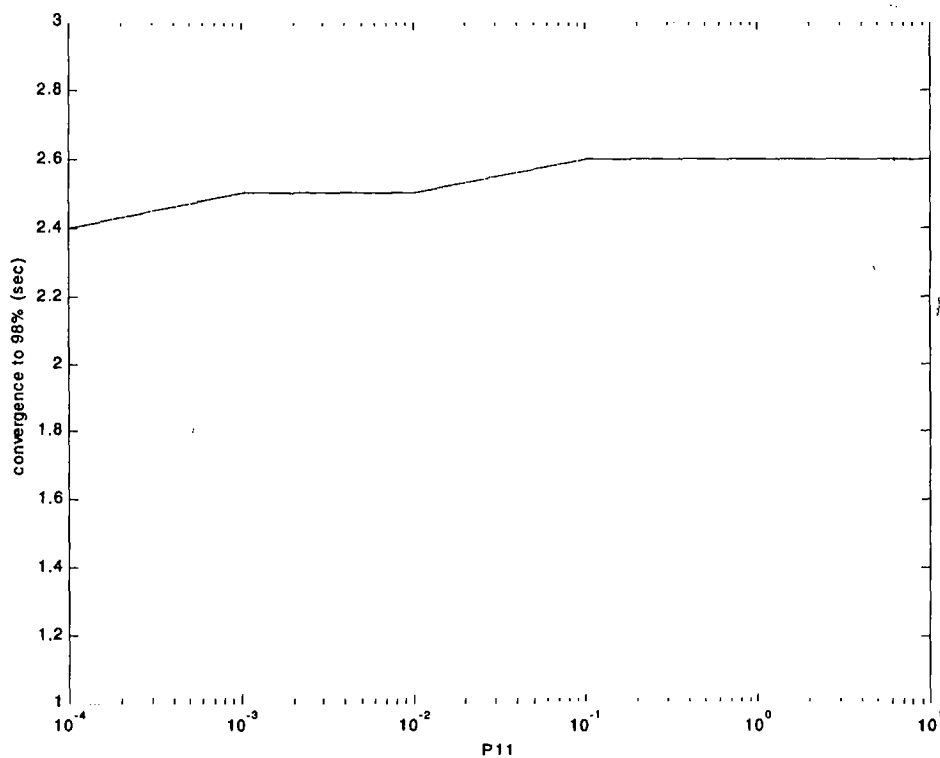


Figure 5. Plot of P_{11} vs. convergence time to within 2% of the actual k

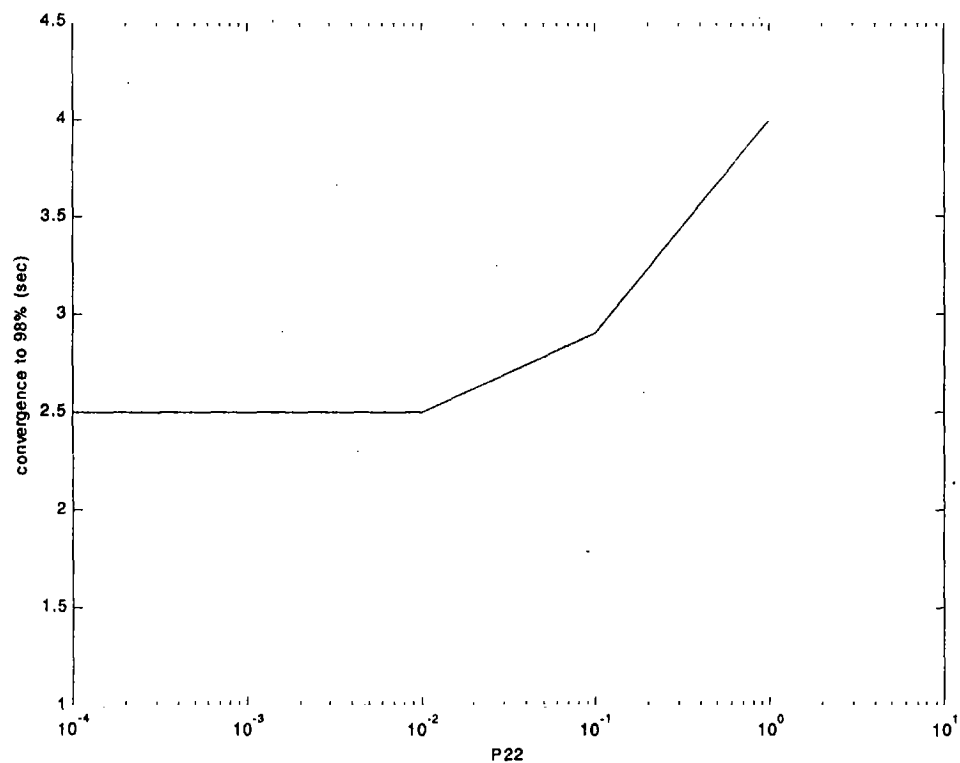


Figure 6. Plot of P_{22} vs. convergence time to within 2% of the actual k value.



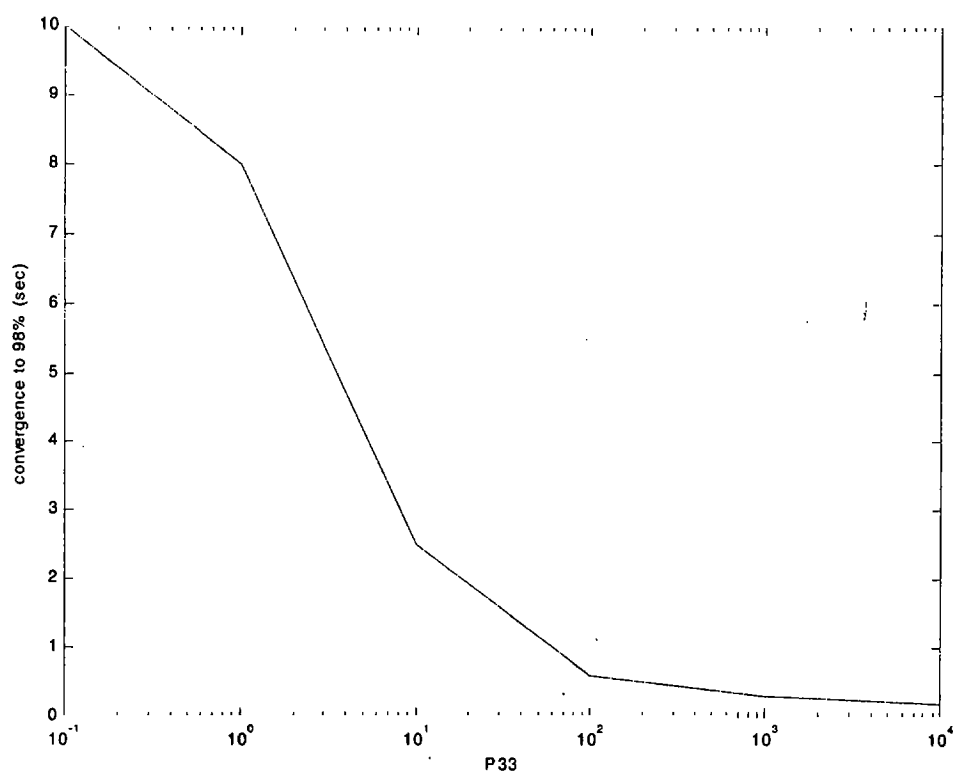


Figure 7. Plot of P_{33} vs. convergence time to within 2% of the actual k value.

Interdepartmental letterhead

Mail Station: L-290

Ext: 33088

1/25/00

To: Dave McCallen
Larry Ng
Dave Harris
Greg Clark

From: Greg Burnett

CC:

Re: The performance of the EKF algorithm compared to the Gauss-Newton for the 10 DOF system

Hello everyone,

In the last few days I have been analyzing the 10 DOF five-story building model with the extended Kalman filter (EKF) as well as looking for ways to make it run better. I wanted to update you on the progress.

To facilitate comparison with the Gauss-Newton (GN) iterative algorithm we used last year, I ran the same battery of tests on it and plotted them on the same plot that I did last year. For the EKF, I used the following values for the noise parameters (as outlined in my last memo "The dependence of convergence time for the EKF on the various noise parameters"):

$R = 0.01 (\pm 0.01 \text{ g's})$ for the measured nodes, 1×10^{12} for the non-measured nodes

$Q_d = P_d = (0.001)^2 = (10\% \text{ of max displacement})^2$

$Q_v = P_v = (0.005)^2 = (10\% \text{ of max velocity})^2$

$Q_k = P_k = (3.1 \times 10^7)^2 = (10\text{x stiffness estimate})^2$

I analyzed 500 samples with a sampling rate of 100 Hz. The driving function was a 1 Hz 1000 amplitude sine wave applied at the first floor. The analyses took about 1.8 hours each on a P2-450 and 2.2 hours on a P2-350.

The results are displayed in Figure 1. The blue lines are the EKF results and the red lines the GN. It is clear that the EKF works much better overall, with the largest error only 15.6% as compared to 179% for the GN. The GN has better precision on the ones it does well, but it must be realized that the EKF is iterative and will likely beat out the GN if given more samples to operate on. Therefore it is fair to say that the EKF outperforms the GN algorithm across the board with largely equivalent computational

University of California



**Lawrence Livermore
National Laboratory**

times. The EKF has the added advantage of being able to use acceleration data directly and is able to operate in real time. It also seems to have a wider “Zone of influence”, being able to converge relatively well with only a single measurement. For the single measurements the following was observed:

Measured node	Error element 1 (%)	Error element 2 (%)	Error element 3 (%)	Error element 4 (%)	Error element 5 (%)	Mean error (%)
Original estimate	-14.5	-33.3	-11.1	-200	-50	61.7
1	4.4	17.9	-1.4	-12.9	-16.7	10.7
2	6.2	-24.6	13.0	-13.3	4.9	12.4
3	2.5	-24.6	11.5	1.8	-34.5	15.0
4	1.9	-23.9	31.4	-6.3	-14.8	15.6
5	-5.8	-22.1	11.2	10.4	17.9	13.5

Table 1. The estimation errors of the original estimate and the EKF algorithm results for a single measurement and 500 samples. The errors in **bold** are the errors for the elements next to the node being observed, the ones in blue are the lowest errors for that particular case and the errors in red are the largest. Note that there is no observed correlation between an observation and the lowest error.

Keep in mind that element 5 is located between nodes 4 and 5, so that an observation of node 1 will yield information on elements 1 and 2 while an observation of only node 5 will only directly observe element 5.

The interesting thing is that the mean errors are all about the same after 500 samples. The algorithm is not terribly sensitive to the location of the largest model mismatches. This is in direct contrast to the GN algorithm in which the largest error was when only node 3 (the most closely modeled node) was observed. The error in that case was 179%, and there was no chance of it getting any lower. The error of the EKF when node 3 was observed was only 15.0%, and with more samples it is possible that the error will decrease.

Another interesting thing was the lack of correlation between measurement and low error rates. Indeed, there is only a single occurrence (node 3 measured) of an element by a measured node having the lowest error. It seems for this simple system that the point of measurement is not a critical parameter.

One last consideration: The input function of this experiment was a sine wave, the convergence may be more rapid and complete with a white noise source. I would like to try this out, but it would take a couple of days and we are short on time. I am proceeding directly to the analysis of the NTS structure, although it might be wiser to start analyzing real data on a smaller structure that can be modeled

Interdepartmental letterhead
 Mail Station: L-290
 Ext: 33088

2/29/00

To: Dave McCallen
 Dave Harris
 Larry Ng
 Jim Candy

From: Greg Burnett

CC:

Re: Estimating model order using the SVD of the generalized Hankel matrix

This is a memo describing the procedures for calculating the model order (if it less than the current estimate) using the Hankel matrix (1). The filename is HANKEL_TEST.M.

The Hankel matrix is represented by

$$H_{rs}(k-1) = \begin{bmatrix} Y_0(k) & Y_0(k+t_1) & \dots & Y_0(k+t_{s-1}) \\ Y_1(k) & Y_1(k+t_1) & \dots & Y_1(k+t_{s-1}) \\ \vdots & \vdots & \vdots & \vdots \\ Y_{j-1}(k) & Y_{j-1}(k+t_1) & \dots & Y_{j-1}(k+t_{s-1}) \end{bmatrix}$$

where $Y_j(k+t)$ is the time response of the system at discrete time t for the j^{th} sensor. It is possible to stack multiple repetitions of an experiment on top of each other to increase the accuracy of the calculation. Then j would be the total number of sensor recordings for all the experiments. For example, if there are 5 sensors and three repetitions, j would equal 15. Once $H(k)$ has been calculated, a singular value decomposition (SVD) is performed such that

$$H = PDQ^T$$

where $D = \text{diag}(d_1, d_2, d_3, \dots, d_n, d_{n+1}, \dots, d_N)$, arranged in ascending order. If H has rank n , all of the singular values d_i ($i = n+1, \dots, N$) should be zero. If some values are not zero but are below a threshold, the number of values above the threshold is determined to be the rank of the system. It can be difficult, however, to determine the threshold in a meaningful manner. The most common ways to do this are to choose a threshold based on the measurement errors and the roundoff errors due to the finite precision of the computer, the latter of which is the most conservative.



Example: 10 DOF system

For the 10 DOF system, the H matrix was calculated using 1, 3 and 10 simulated 200-sample responses. White noise and a 1 Hz sine wave were used as the inputs. For the sine wave, the SV were:

d =	d =	d =
1.0194	1.4417	3.2236
0.3159	0.44675	0.99897
0.17534	0.24796	0.55446
0.098458	0.13924	0.31135
0.038704	0.054736	0.12239
0.0027782	0.0039289	0.0087854
1.5002e-007	2.1216e-007	4.7442e-007
9.1443e-010	1.2932e-009	2.8917e-009
3.6045e-013	5.0976e-013	1.1399e-012
7.1239e-017	1.3691e-016	3.4406e-016
	9.2406e-017	3.1178e-016
	5.4194e-017	9.181e-017
	1.0246e-017	6.7115e-017
	3.1416e-018	2.6311e-024
	.	.
	.	.
	.	.
	9.859e-026	5.82e-152
	5.2552e-026	1.7477e-154
	0	0

Table 1. Singular values for 10 DOF with 1, 3, and 10 sine wave experiments

In Table 2, the results using “white” noise (it’s not terribly white) as the input were:

d =	d =	d =
3.0879	5.7475	10.829
1.8621	3.6117	5.5977
1.1926	2.0775	3.5007
0.55992	1.7958	3.2094
0.30377	0.50934	0.84204
0.0060326	0.011384	0.017797
3.3942e-006	5.501e-006	1.2048e-005
5.9874e-009	2.8399e-009	3.5921e-008
4.0753e-012	6.5487e-013	9.1223e-012
1.6622e-016	5.4572e-016	1.3807e-015
	3.8749e-016	1.1974e-015
	2.4835e-016	6.747e-016
	1.2985e-016	5.1779e-016
	1.0114e-016	4.1357e-016
	.	.
	.	.
	.	.
	1.7495e-041	7.9383e-151
	3.8106e-045	8.2083e-156
	3.1696e-048	2.9522e-158

Table 2. Singular values for 10 DOF with 1, 3, and 10 white noise experiments

Using the machine precision of about 1×10^{-16} as our threshold (a conservative one), it is clear that this is about a 10 DOF system. Less conservatively, using 0.01% of the maximum acceleration measured as a threshold (on the order of 1×10^{-6}), we can describe the system only 6 DOF and still be able to model the process adequately. Thus the system may be slightly over-specified, but not significantly.

If we repeat the effort using the 50 DOF system (using only 100 samples to save time), we see in Table 3 that the 50 DOF system can be described quite well using the 0.01% threshold (in this case about 10^{-3}) with only about 13 DOF. This confirms that the system is substantially over-specified, and this may be causing problems in the identification algorithm.

NTS 5 DOF:

Just to see what happens, let's try the analysis on the NTS 5 DOF system. I used 2 seconds of data (800 samples) from each experiment. After piecing together the data from five of the white noise experiments and 1 of the swept sine, we see in Table 4 that the singular values are very large, much



5.4517e+005	9.9645e+005	7.5627e+005
3.7771	7.4706	0.81524
0.31248	0.63342	0.49348
0.20186	0.44984	0.12125
0.19007	0.41857	0.088409
0.1405	0.29166	0.069179
0.084585	0.14801	0.051343
0.067245	0.08745	0.027197
0.038537	0.059135	0.010946
0.0053582	0.021721	0.009895
0.0022263	0.01341	0.0030746
0.0013009	0.006996	0.00076346
0.0011309	0.0030913	0.00038853
8.7946e-005	0.00015612	0.00017931
4.5737e-005	9.2437e-005	7.543e-005
3.1967e-006	6.9722e-006	6.6894e-006
3.6162e-007	2.4413e-006	1.1186e-006
1.3889e-007	1.3383e-006	5.2704e-007
1.0214e-007	2.455e-007	2.8841e-007
4.552e-008	1.0369e-007	8.9292e-008
2.9745e-008	6.259e-008	1.6371e-008
7.7709e-009	1.862e-008	4.5668e-009
3.4277e-009	5.119e-009	2.6309e-009
2.5221e-010	7.8879e-010	6.6511e-010
1.6772e-010	5.2503e-010	1.5055e-010
9.1908e-011	1.9397e-010	8.8564e-011
8.7112e-011	7.5341e-011	3.5151e-011
1.8675e-011	5.9365e-011	2.4732e-011
1.6054e-011	4.1318e-011	1.7685e-011
5.2008e-012	3.0092e-011	1.5558e-011
4.2227e-012	2.3498e-011	8.5962e-012
1.111e-012	5.1674e-012	4.1185e-012
6.7755e-013	1.6663e-012	2.6765e-012
4.3064e-013	9.5748e-013	7.105e-013
2.1113e-013	8.7482e-013	6.0978e-013
1.6548e-013	5.3575e-013	4.6454e-013
9.0577e-014	1.8523e-013	3.1747e-013
5.7861e-014	1.7951e-013	3.0546e-013
5.1306e-014	9.7289e-014	2.1142e-013
4.1158e-014	6.6973e-014	1.7061e-013
3.6638e-014	6.2035e-014	1.6e-013
3.1338e-014	6.0726e-014	1.2708e-013
2.6422e-014	5.5899e-014	9.6045e-014
2.2437e-014	3.6316e-014	8.8803e-014
1.7044e-014	3.1304e-014	7.2361e-014
1.36e-014	.	.
1.2065e-014	.	.
3.788e-018	.	.
9.1338e-019	9.6218e-032	2.9283e-034
7.4538e-019	2.1657e-033	0

Table 3. Singular values for 50 DOF with 1 and 3 trials with white noise input and 3 trials with sine wave inputs



2250.2	15.034
1163.3	11.737
1082.6	9.5923
848.84	8.1228
663.6	7.4599
585.06	
461.33	
392.48	
262.89	
238.58	
210.16	
164.19	
134.88	
94.636	
88.062	
69.076	
64.416	
56.636	
47.742	
38.71	
38.205	
29.843	
25.403	
21.567	
19.645	

Table 4. Singular values for the NTS 5 DOF model with 5 white noise excitations and 1 swept sine excitation.

larger than the thresholds defined by machine precision or 0.01% of the maximum acceleration measured (about 10^{-4}). Therefore we may conclude that the model requires more than five DOF to describe the system adequately.

References: [1] Juang, J.N. and Pappa, R.S. (1985). "An eigensystem realization algorithm for model parameter identification and model reduction", *J. Guidance Control Dyn.*, V. 8, (5), pp. 620-627.



Interdepartmental letterhead

Mail Station: L-290

Ext: 33088

3/1/00

To: Dave McCallen
Greg Clark
Dave Harris
Jim Candy

From: Greg Burnett

CC:

Re: Extended Kalman filter results for 10 DOF, 50 DOF, and NTS 5 DOF

Hello everyone,


This is a memo detailing the results I have compiled by testing the Extended Kalman Filter (EKF, see my earlier memo "The performance of the EKF algorithm compared to the Gauss-Newton for the 10 DOF system" of 1/25/00 and "Augmented state vector continuous-discrete extended Kalman filter system identification approach" of 12/22/99 for details on the algorithm) with simulated 10 and 50 DOF 5 story building as well as the real NTS structure, modeled as a 5 DOF system.

10 DOF with and without noise

Here, the 10 DOF (5 translational and 5 rotational) model was tested with different levels of noise and with a variety of measured and unmeasured nodes. The rotational nodes were not considered measured, and the number of translational nodes measured was varied. The unmeasured nodes were assigned a variance of 1×10^{12} , effectively rendering each "measurement" completely ineffective in updating the state. The results for no added noise and a white noise input were compiled in my last memo of 1/25/00. To summarize, the EKF could identify the system quite well for just about any combination of measurements, including only 1 or 2 measurements. It was much more precise and robust than the Gauss-Newton iterative search. However, it was shown that it was quite dependent on a complete calculation of the **P** propagation, which can take a significant amount of resources to compute. Also, the effects of changing the input function or the noise inputs and matrices was not addressed.

To continue the examination of the performance of the EKF for the 10 DOF system, white noise with different S/N ratios was added to the simulated measurements and different input functions and values for the measurement covariance matrix **R** were used. The white noise was added at levels of -20 and -10 dB S/N. For example, for a S/N of -20 dB, white noise with a maximum amplitude of 1/10 of the maximum of the corresponding clean measurement was added to each measurement. The noise was

University of California

 **Lawrence Livermore
National Laboratory**

therefore different for each DOF. A noise level of -20 dB is a significant amount of noise for sensitive accelerometers and should be a good indication of noise robustness. The measurement noise covariance was approximated by a constant for early experiments; in later trials the actual covariance of the noise added to the measurements was used.

Experiment setup

There are several degrees of freedom here, namely:

- I. The type of excitation (white noise, sine wave)
- II. The level of noise added to the signal (none, -20 dB, -10 dB)
- III. The size and composition of the \mathbf{R} (measurement covariance) matrix
 - a. Same for all nodes (0.01, 0.1)
 - b. Different for each node (measured for each one)
- IV. Which nodes are observed (all, some, one)

I therefore ran several trials, varying one parameter at a time to come up with the best performance. My standard perturbation of the initial parameter estimate * [7/8 3/4 9/10 1/3 2/3] will be used. That is, the actual structure will have k values that are 7/8, 3/4, 9/10, 1/3, and 2/3 of the estimate. The experiments will be performed in the following order:

1. White noise input, no noise added, $\mathbf{R} = \text{diag}(0.01)$ (lower bound)
2. White noise input, no noise added, $\mathbf{R} = \text{diag}(0.1)$ (upper bound)
3. White noise input, no noise added, $\mathbf{R} = \text{measured for each node}$

Of these experiments, trial 2 yielded the best results. Therefore I have fixed \mathbf{R} at $\text{diag}(0.1)$ for the rest of the experiments. It is not as good for some no-noise situations, but it is essential to only change one variable at a time. The other two experiments with white noise inputs were

4. White noise input, noise added at -20 dB, $\mathbf{R} = \text{diag}(0.1)$
5. White noise input, noise added at -10 dB, $\mathbf{R} = \text{diag}(0.1)$

This should give us a good idea of the ability of the algorithm to converge in the presence of noise.

Incidentally, when using a white noise input the performance of the algorithm would fluctuate, sometimes significantly, so some of the above results are averages over several experiments. These fluctuations are not present when a sine wave input is used. I don't know at the present time why the performance would vary so much with the white noise and not the sine wave, which is only a single frequency and shouldn't excite the structure as well.



I now repeat the above using a sine wave input, with frequency 1 Hz.

6. Sine wave input, no noise added, $\mathbf{R} = \text{diag}(0.01)$ (lower bound)
7. Sine wave input, no noise added, $\mathbf{R} = \text{diag}(0.1)$ (upper bound)
8. Sine wave input, no noise added, $\mathbf{R} = \text{measured for each node}$
9. Sine wave input, noise added at -20 dB, $\mathbf{R} = \text{diag}(0.1)$
10. Sine wave input, noise added at -10 dB, $\mathbf{R} = \text{diag}(0.1)$

Again, using $\mathbf{R} = \text{diag}(0.1)$ led to the lowest error rates. The results for two separate iterations are shown in Figures 1 and 2. The mean identification errors when the last 100 estimates are averaged are shown in blue, and the single last estimate is shown in red. This can help us see where there is still some oscillation around the correct answer. It is clear from the differences in Figures 1 and 2 that the accuracy of the identification can vary significantly if the white noise input is used. The sine wave input, on the other hand, yields identical answers when there is no added white noise (as expected) and exhibits only a slight variation when white noise is added to the measurements. I do not know why the performance of the algorithm varies so widely when white noise is used as the input, but it makes it difficult to compare performance. Therefore, for the observability test the sine wave input shall be used, as it results in a more predictable performance.

Now we may examine the effects of observability. The same parameter perturbation as above was used, while the number and locations of the observations were varied. The results are shown in Figure 3. In this figure, each set of stacked boxes As before (see memo of 1/25/00, "The performance of the EKF algorithm..."), the EKF (blue bars) performed reasonably well over the entire structure, unlike the Gauss-Newton algorithm (red bars). Even with a single measurement reasonable accuracy is obtained.

Conclusion for 10 DOF

The EKF works quite well on the 10 DOF system, with and without noise present. The next test should be combining a low number of measurements with added noise to see if identification is still possible with only one or two sensors in the presence of noise.

50 DOF without noise, only floors measured

This model is the same as the 10 DOF with the column discretized into 9 sub-columns. The measurements are assumed to take place only at the floors, so the number of measurements stays the same at 5 but the number of DOF goes up by a factor of 5.

I have not yet been able to make any progress on the 50 DOF model of the five-story building. I have varied the noise covariance matrices, the input amplitude, the size of the perturbation, and the location and size of the perturbations, and I can get no convergence at all. Worse, the algorithm is very slow – about 50 hours for 100 samples. This is due to the very large size of P : 125×125 . This means there are 7750 independent members of P that have to be propagated, so ode45 has to be run 7750 times per time sample. This is quite costly.

However, even with the high cost we still have no positive results. The nodes that are perturbed are not detected, even with a change of up to 50%, and nodes far away from the damage are sometimes identified with errors ranging up to 50%. With the same amplitude input function as used for the 10 DOF, the changes in the output due to the perturbations (even for 20% changes in stiffness) were very small. It was necessary to multiply the input by a factor of 1000 in order to make the errors large enough to affect the output, something that should not be necessary as the 10 and 50 DOF systems are very similar models. The estimation results for 5 of the 25 of the stiffness values (these are ones located next to the floor nodes) are shown in Figure 4. It is clear that there is little convergence, except possibly for states 110 and 115. The other three estimates are very poor and are headed in the wrong direction. It is possible that a larger number of samples would yield better results, but as it currently takes an hour to compute two samples, further computations have not been attempted yet. It is also possible that we are suffering the same problems as before using a white noise input, but I have seen the same results five times in a row, making the possibility remote. I am re-running the experiment with a sine wave input just to be sure, though.

I have an idea as to why identification of this system has been difficult. This is a model that can be completely specified by 10 DOF, yet we are attempting to describe it using 50 DOF. Thus we have the classic problem of over-specification, where many of the DOF are not necessary to describe the system. This may be the reason we are having trouble identifying it. It is also true that over-specification is the exact opposite of the situation we will commonly encounter in practice, where our model will most likely under-specify the system. Thus it may not be the best test of the algorithm. I would suggest a different model, one that requires 50 or more DOF in order to completely describe the system. This may be the only way to effectively test the algorithm. We will still be limited by the long computational times, but at least we will be able to more effectively test the algorithm. I have used the SVD of the generalized Hankel matrix to estimate the order of the matrices for the 10, 50, and NTS 5 DOF systems. The results confirmed that only at most 14 or so DOF is needed for the 50 DOF system (see “Estimating model order using the SVD of the generalized Hankel matrix” of 2/29/00).

NTS 5 DOF

This is a 5 DOF model that describes the large 5-story model instrumented at NTS. I believe this model has the opposite problem as the one described in the 50 DOF section, as it is far too low order to

describe the system effectively. The SVD of the Hankel matrix confirmed that the system is under-specified. I have tried many different values for the noise matrices P , Q and R , and have not been able to get the system to converge, even for up to 59 seconds of data at 400 samples/second. The stiffness values do not change much at all or just continue to change very slowly, with no hint of convergence. At Dave Mc's suggestion, I even tried changing the algorithm so that every independent value of K would become a parameter, in order to ensure that the algorithm had the most flexibility possible. This worked on the 10 DOF, but not on this system.

I have proposed to Dave Mc that we first try to identify a simpler known system, such as coupled oscillators with three or more masses. This would allow us to test the algorithm with real data, yet the system would be well understood and easily described with only a few DOF. I would also like to be able to add a fourth, smaller mass to the system to explore to what extent model inconsistencies may be accounted for in the designation of Q , the system propagation noise matrix. The nature of the EKF is that some modeling inaccuracies may be compensated for by enlarging Q , but how large the inaccuracies may be and the amount of Q change required is not clear. It would be nice to be able to describe this quantitatively. This experiment would also make a good paper.

Conclusions

I have been able to show that the EKF works quite well with the 10 DOF simulated system, even with -10 dB added noise in the measurements. However, this is a simulated system, and the modeled and actual systems are the same order – indeed, they are identical except for the stiffness values. It may be interesting to change the experiment slightly so that the model is only 8 DOF, and then see if we can still estimate the system well. It may also be useful to process the measurement data to get an idea of what order model should be used to simulate the system. There are several ways to do this, using singular value decomposition and others, and it may be useful to look into that. However, for this year, we may want to limit ourselves to understanding the limits under which the EKF and its like may operate successfully.

I also believe that the 50 DOF model should be revamped to represent a system that requires close to 50 DOF to be described correctly. As for the NTS structure, I think we should start a little smaller and instrument a real low DOF system that we can describe using a simple model. That way we can wring out the algorithms and the noise problems more systematically.



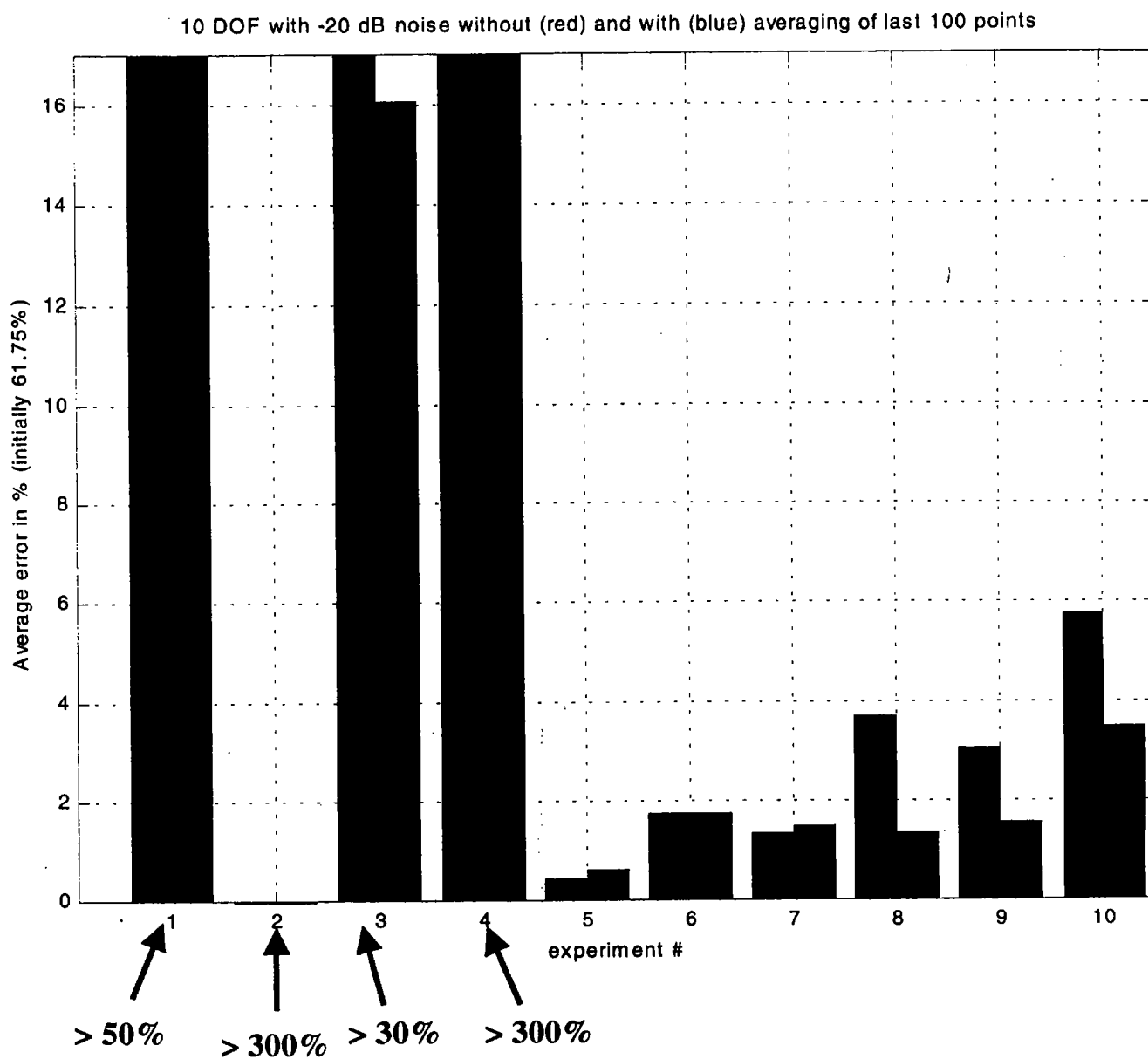


Figure 1. Outcomes for the 10 experiments above, first iteration. The left (red) bar represents the mean identification error with no averaging, while the right (blue) bar is the error with the last 100 samples averaged. It is clear that the sine wave input results in a more accurate identification.

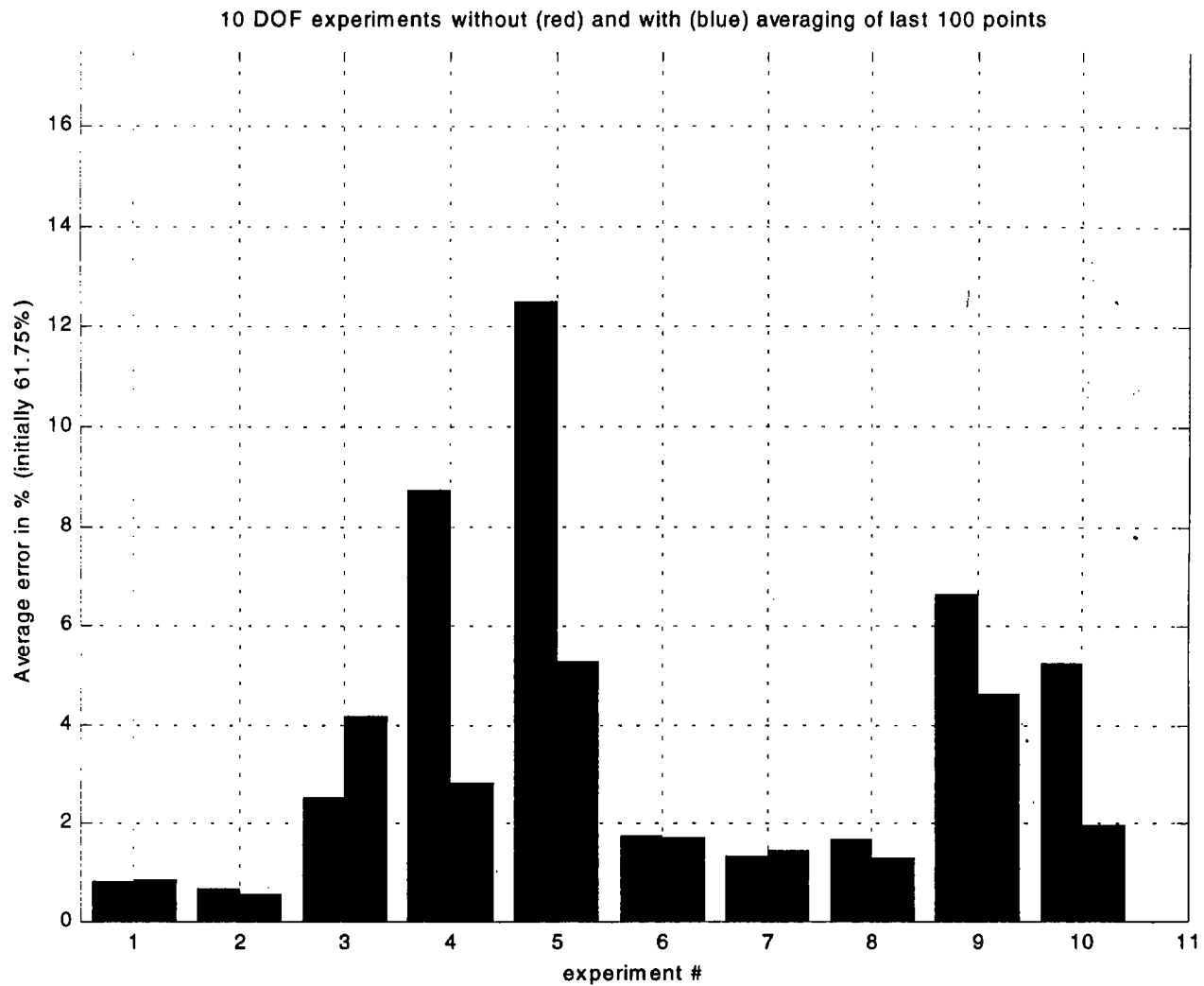


Figure 2. Outcomes for the 10 experiments above, second iteration. The left (red) bar represents the mean identification error with no averaging, while the right (blue) bar is the error with the last 100 samples averaged. It is clear that the white noise input is much better behaved in this example, although overall the sine wave input still results in a more accurate identification.

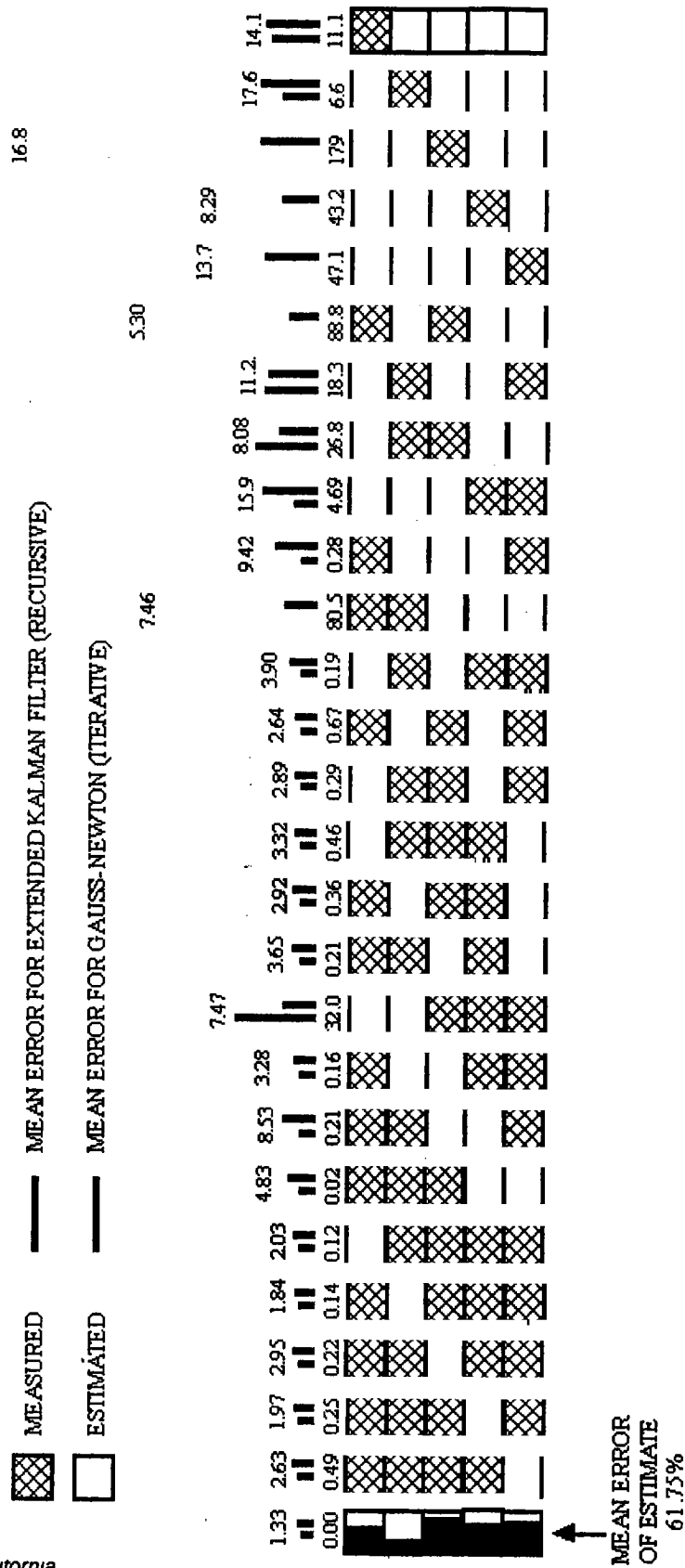


Figure 3. Mean identification error for 10 DOF using Gauss-Newton method (red bars on left) and the EKF with sine input, $R = 0.1$, no added noise (blue bars)

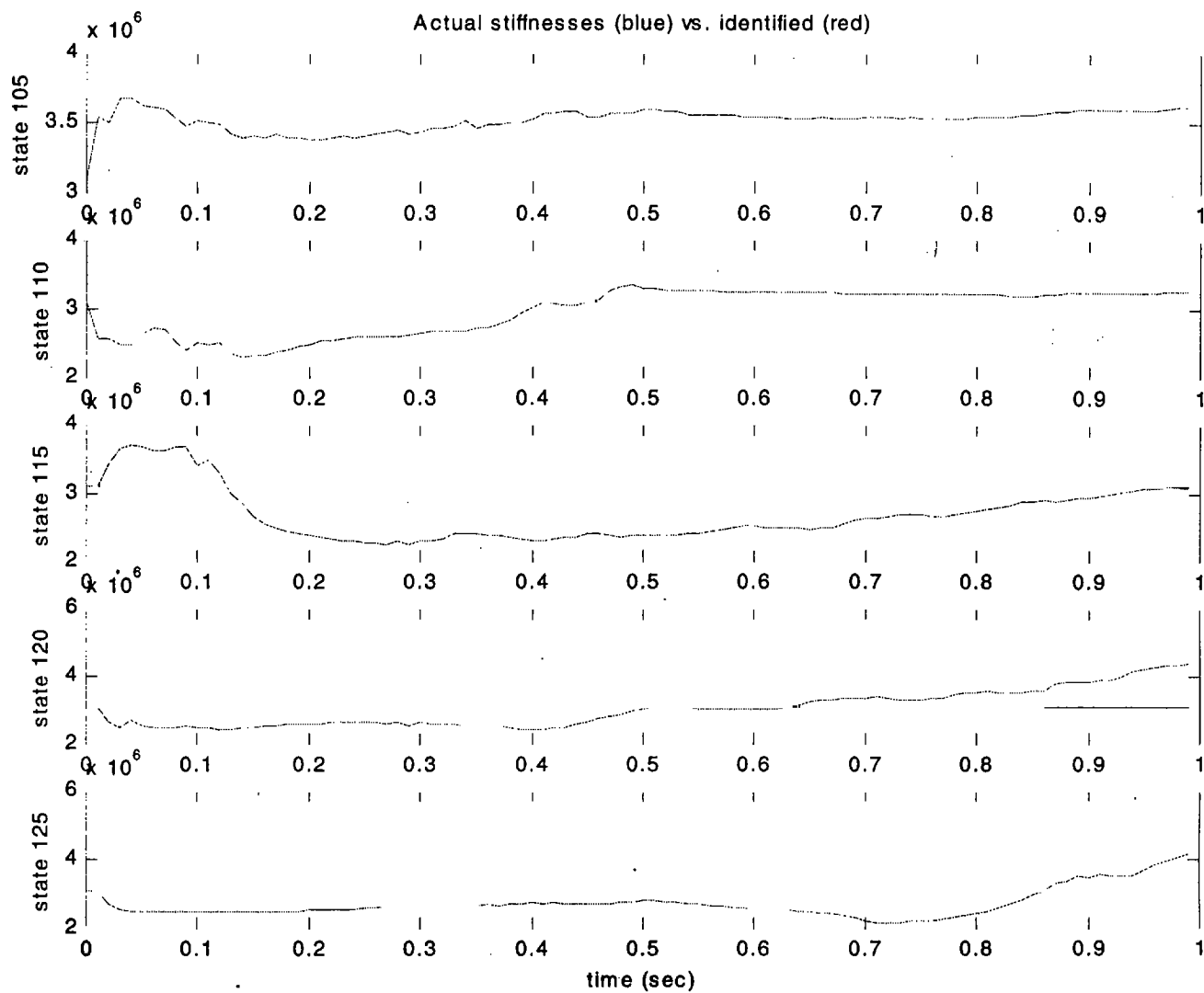


Figure 4. The actual values for five of the 25 stiffness value to be identified (blue), and the estimated values (red). These five values are next to observed nodes and should be easily identified. Note that the y axis does not the same scale for every plot.

Interdepartmental letterhead
Mail Station: L-271
Ext: 33088

3/1/00

To: Dave McCallen
Dave Harris
Jim Candy
Larry Ng

From: Greg Burnett

CC:

Re: Suggestions and comments from advisory meeting of 3/1/00

Hello everyone,

Thank you to everyone who could attend the meeting today. I appreciate you all coming, it's nice once in a while to be able to mine the gold of the Lab. ☺

Here's a list of observations, suggestions, and comments from the meeting:

10 DOF, 50 DOF, General

1. Use the more stable Joseph form for calculating the updated state covariance **P**.
2. Use a pseudorandom input to the system, it has a wide bandwidth but is deterministic and repeatable. Using a "random" (but known) input does not lead to repeatable results as it is but a single realization of a random process. Using a sine wave is not effective at exciting many modes of the system.
3. Since the 50 DOF system seems to be highly over-specified, try simulating 5 measurements given the 50 DOF model and then try to identify it using the 10 DOF model. This will help determine if the over-specification is causing convergence problems.
4. Add an insignificant amount (i.e. $1e^{-12}$) of noise to the data for "noise-free" simulations to avoid singular matrices in the algorithm.
5. Check the 50 DOF residuals to see if they are white even though the algorithm has not converged to an answer. If they are, then there is no way the EKF can calculate a solution.



6. Since $\hat{\mathbf{x}}[k_+] = \hat{\mathbf{x}}[k_-] + \mathbf{K} \cdot \mathbf{e}$, check for $\Delta \mathbf{K}$ to drop below a threshold value. Once it does, replace it with a constant – this will speed up processing.
7. Examine the \mathbf{P} propagation in more detail. Since

$$\dot{\mathbf{P}}(t) \approx \mathbf{F}(\hat{\mathbf{x}}, t)\mathbf{P}(t) + \mathbf{P}(t)\mathbf{F}^T(\hat{\mathbf{x}}, t) + \mathbf{Q}(t)$$
 and we know a lot about \mathbf{F} and \mathbf{Q} , it may be possible to speed up the propagation calculation by excluding some parts of the right side of the above equation from the ODE solver.
8. Check $\Delta \mathbf{P}$, so that if parts of it get very small, they may be excluded from the ODE calculation.
9. See if the \mathbf{P} propagation can be split into parts, some of which are simple enough to do with an exponential matrix calculation.

NTS structure

1. Use the MUSIC (Multiple Signal Classification) algorithm to determine the pole locations given the measured output of the structure.
2. Do the same thing to the innovation sequences. This will determine the location of the uncaptured modes of the system. Can also do it by comparing the spectra of the model and measured system.
3. Bandpass filter the data so that the modes the model is not able to capture are excluded from the data.
4. Use a simple grid-search or similar method to calculate the initial estimates of the stiffness parameters. Then run the EKF with the new estimates.

I think that's about it. If I have missed anything or if you have any further suggestions let me know.
Thanks again for coming!

Greg

Interdepartmental letterhead

Mail Station: L-271

Ext: 33088

4/28/00

To: Dave McCallen
Dave Harris
Jim Candy
Larry Ng

From: Greg Burnett

CC:

Re: Action on suggestions and comments from advisory meeting of 3/1/00

Hello everyone,

This is a report detailing the actions taken on the suggestions received at the advisory meeting of March 1. I have listed the action after each recommendation.

10 DOF, 50 DOF, General

- 1. Use the more stable Joseph form for calculating the updated state covariance P.**

Action: Implemented.

- 2. Use a pseudorandom input to the system, it has a wide bandwidth but is deterministic and repeatable. Using a "random" (but known) input does not lead to repeatable results as it is but a single realization of a random process. Using a sine wave is not effective at exciting many modes of the system.**

Action: Used IDINPUT.M to build a pseudorandom input function from sine waves that has many of the qualities of white noise over a specified bandwidth. Algorithm performance now much more repeatable.

- 3. Since the 50 DOF system seems to be highly over-specified, try simulating 5 measurements given the 50 DOF model and then try to identify it using the 10 DOF model. This will help determine if the over-specification is causing convergence problems.**

Action: Implemented, and the 10 DOF model identified the 50 DOF system rather well, converging to an error of about 5-10 % within about 20 samples. True convergence was never achieved, as shown in Figure 1, but the identified values oscillated around the correct ones and the average error (taken over a few hundred samples) was clearly quite small. This would seem to indicate that the 50 DOF system may not be identifiable using a 50 DOF model as this is a vast overspecification of the system in question. This agrees with the results from my memo of 2/29/00, "Estimating model order using the SVD of the Generalized Hankel Matrix".

4. Add an insignificant amount (i.e. $1e^{-12}$ or -240 dB) of noise to the data for "noise-free" simulations to avoid singular matrices in the algorithm.

Action: This was implemented, and a reduction in the maximum of ΔK , ΔX , and ΔP by about three orders of magnitude was observed. Clearly the algorithm is more stable with this option. The practice will be continued in the future to avoid any potential singularity problems.

5. Check the 50 DOF residuals to see if they are white even though the algorithm has not converged to an answer. If they are, then there is no way the EKF can calculate a solution.

Action: The 50 DOF algorithm takes about 25 minutes to calculate each residual point, so it would take a significant time to generate enough points for a meaningful analysis. As the results of Section 3 have shown the system to be very overspecified, it is probably not worth the time to complete this experiment. It is almost certain that most of the residuals would be white, as most of the DOF are superfluous and have little effect on the output of the system.

6. Since $\hat{x}[k_+] = \hat{x}[k_-] + K \cdot e$, check for ΔK to drop below a threshold value. Once it does, replace it with a constant – this will speed up processing.

Action: The 10 DOF system was run with no noise, a standard (large) perturbation, and the pseudorandom input. The changes in K , P , the states X , and the innovations (y -yhat) were calculated at each time step. This was done to see if K and P would converge to some sort of steady-state value. The results for K are plotted in Figure 2. Here, the average change (in percent) for all the member of K is shown for each time step. Clearly, K did not settle rapidly to a steady-state value, instead there were significant changes in K long after the algorithm had settled on the correct stiffness values. This was true with and without adding the small amount of noise to the system as discussed in Section 4. The algorithm had converged to within about 2% accuracy by 75 samples, but there are still very large changes in K evident long after convergence occurred (red arrow). This shows that in this case, setting K constant after a short period of little change could have significant impact on the ability of the algorithm to converge to the correct answer.

As an example, I tried using a ΔK threshold of 10% (after the change in K drops below 10%, it was kept constant) to observe the effect. In the original algorithm, the identification error rate was less than 1% and it took less than 6 minutes to converge at 80 samples. Using a ΔK threshold of 10%, the identification error jumped to 81.4% and convergence was not attained, as the accuracy was not good enough to stop the processing (the changes in the stiffness values are examined, once they drop below a threshold 20 times in a row the iterations are terminated). In this particular case ΔK dropped below 10% and was held constant after only 7 samples, and although the algorithm ran in about one-half of the normal time, the identification error was not acceptable. Smaller values of ΔK were tried, but none resulted in reliable identification accuracy and reduced computational times. Therefore ΔK is not at present seen as a way to decrease the computational expense of the algorithm.

The present system uses a subroutine that checks to see when the changes in the stiffness states of X drop below about 0.2% for 20 subsequent samples and it seems to work pretty well. It terminated the iterations for the experiment above after only 80 samples, and was accurate to less than 1 percent.

However, I wanted to check and see if all of ΔX (including the displacement and velocity states, not just the stiffness states) would be a better termination criterion than just the stiffness states. Figures 3 and 4 show the change in X (all states) and the change in the X (stiffness states only) vs. sample for the 10 DOF model. It is clear that using all states, ΔX is not that stable, with relatively large changes occurring long after the convergence time of 75 samples. However, the change in X using stiffness states only is very good, with the change in percent reaching essentially zero by 75 samples. This is what I have been using to limit the number of iterations needed, and it works quite well in that capacity.

Thus my conclusion here is that the change in K is not a reliable indicator of algorithm convergence, rather the change in the identified parameters is a better termination criterion.

7. Examine the P propagation in more detail. Since

$$\dot{P}(t) \approx F(\hat{x}, t)P(t) + P(t)F^T(\hat{x}, t) + Q(t)$$

and we know a lot about F and Q , it may be possible to speed up the propagation calculation by excluding some parts of the right side of the above equation from the ODE solver.

Action: The P propagation was examined, and it was determined that there was no simpler way to calculate the above relation. For example, in a 1-D system, the F matrix is

$$\mathbf{F} = \begin{bmatrix} 0 & 1 & 1 \\ -x_3/m & -b/m & -x_1/m \\ 0 & 0 & 0 \end{bmatrix}$$

where x_1 is the displacement, x_2 is the velocity, and x_3 is the stiffness parameter (see memo "Augmented state vector continuous-discrete extended Kalman filter system identification approach" of 12/22/99). Even if \mathbf{P} starts off as diagonal, it rapidly fills, and by taking first $\mathbf{F}\mathbf{P}$ then $\mathbf{P}\mathbf{F}^T$ we get a thorough mixing of variables such that the equation for $\dot{\mathbf{P}}$ is not a simple one, even without the addition of \mathbf{Q} . Thus there is no shortcut that can be taken in the calculation of $\dot{\mathbf{P}}$.

8. Check $\Delta\mathbf{P}$, so that if parts of it get very small, they may be excluded from the ODE calculation.

Action: I have tried a few techniques in the past with no success in this area. These include using a linear and quadratic interpolation for $\dot{\mathbf{P}}$, using a constant \mathbf{P} after $\Delta\mathbf{P}$ dropped below a threshold, and compiling parts of the $\dot{\mathbf{P}}$ calculation. I tried four techniques this time. The first was as above in Section 6, where I looked at the average change in percent of \mathbf{P} from one sample to the next. This was to determine if \mathbf{P} converged to a steady state value and could therefore be expressed as a constant after some time. This technique did not work, but I wanted to know why so I plotted the changes in \mathbf{P} vs. time.

The results are shown in Figure 5. It is clear that like \mathbf{K} , the members of \mathbf{P} do not converge to some steady-state value; there is still significant change after 75 samples, the convergence time. Thus it will not be possible to use a constant \mathbf{P} at any time to save computational cycles.

The second technique considered was to take the difference of *each member* of \mathbf{P} just before and just after the update calculation. If the difference was below a threshold, *that member* of \mathbf{P} was held constant and not propagated using ODE45.M, saving comp cycles. This differs from the technique above, in which the entire \mathbf{P} was held constant if the mean change was below a threshold. Each member was tested each time, so that a member could be held constant one round and then be propagated the next. This resulted in significant computational overhead, but I wanted to see if there was a threshold we could use that would save enough time to justify the extra computational expense.

The results from the second test are more involved and are tabulated in the first 11 rows of Table 1. Some experiments were run more than once to get an idea of the variability of the results. The criterion for convergence was a threshold in the change in stiffness states, detailed above. It is clear that a $\Delta\mathbf{P}_i$ threshold of about 1% was the maximum that could be used without raising the error rate to unacceptable levels. Unfortunately, this threshold did not result in a reduction in computation

time, even though almost 5% of the ODE calculations were skipped. It is thought that the overhead associated with keeping track of the changes in \mathbf{P} probably results in little computational savings. Also, the reduced accuracy due to the approximation in \mathbf{P} means that the algorithm does not converge as quickly to the correct values, somewhat increasing the calculation time. Therefore this option does not seem to be a wise one – approximating changes in \mathbf{P} results does not significantly reduce computational expense.

The third method was to limit the area of \mathbf{P} that was calculated each time. Since the stiffness states are the only ones of interest, it may be that the \mathbf{P} calculation is only critical where the stiffness states are concerned. Perhaps it would be possible to set the \mathbf{P} calc constant for the displacement and velocity parts and only calculate those parts of \mathbf{P} related to the stiffness states. The result is shown in row 12 of Table 1. It actually seems to work pretty well, enabling a reduction in calls to ODE45.M by over 95% (reducing the runtime by over 50%) and still getting pretty good stiffness estimates (8.2% error) by averaging the last two hundred points. By plotting the estimates, it was clear that they oscillated about the actual stiffness values, indicating weak convergence. This warranted further exploration. A compromise was tried in which all of \mathbf{P} is calculated every 5th or 10th step and the stiffness state propagation is calculated the rest of the time (every sample). The results are shown in Table 1 in rows 13 and 14. It worked surprisingly well, achieving a mean identification error of only 1.16% (1.39% with the last two hundred estimates averaged) for every 5th calculation to 3.35% (1.97%) for every 10th. At the same time, it took only slightly longer than the threshold of 1% and about ½ the time of the 5% $\Delta\mathbf{P}_i$ threshold.

The fourth method is quite simple – just calculate $\dot{\mathbf{P}}$ every 5th or 10th time and consider it constant the rest of the time, skipping the stiffness updates. This worked quite well, with 80% of the ODE45.M calculations skipped and identification error rates of only 0.39% calculating every 5th $\dot{\mathbf{P}}$ and 1.97% every 10th. This accuracy held up for a variety of inputs. It seems the stiffness states need not be calculated each time to maintain a decent level of accuracy. As the run times were essentially the same, calculating $\dot{\mathbf{P}}$ every 5th sample seems to be a good choice. I think this is the best way to achieve reasonable identification accuracy while saving some computational expense. In hindsight, this should have been attempted first, as it is the simplest of all the ones I have tried. Of course, if everything went perfectly it would still be faster to calculate $\dot{\mathbf{P}}$ every time, as the convergence is faster. But for situations where convergence is not rapid or assured, only calculating $\dot{\mathbf{P}}$ every 5th step may make sense.

9. See if the $\dot{\mathbf{P}}$ calculation can be split into parts, some of which are simple enough to do with an exponential matrix calculation.

Action: Due to the addition of $\mathbf{F}\mathbf{P}$ and $\mathbf{P}\mathbf{F}^T$, there is no simple split that can be made to simplify the calculation.

NTS structure

1. Use the MUSIC (Multiple Signal Classification) algorithm to determine the pole locations given the measured output of the structure.

Action: Completed, results for STRUCOW5.MAT (undamaged structure excited with a white noise input) shown in Figure 6. From the MUSIC algorithm (PMUSIC.M) it would seem that the resonances of the system are located at about 2.0, 14.2, 32.0, and 37.6 Hz (the resonances near 60 Hz are probably just noise). However, a glance at the power spectral densities (psds) of each channel (Figure 7) reveal that only the first floor has a resonance at 2.0 Hz, the rest are all at 4.4, 14.4, 32.4, and 38.0 Hz, which seem more reasonable. I am not sure why MUSIC places the lowest resonance at 2.0 Hz rather than 4.4, as only one channel exhibits that response and four others do not. What is clear is that there is a significant amount of energy near DC for all the sensors, indicating a substantial DC drift, which was observed and filtered out before processing.

Both methods show that there is significant energy above 50 Hz in the output of the sensors that is probably not due to the input of the system, which was supposed to be bandlimited to 50 Hz but had a 3-dB frequency of about 55 Hz and was only about 10 dB down at 60 Hz. Thus this energy may be due in part to input frequencies at a decreased level, system nonlinearities and noise. This energy was previously noticed and removed before identification was attempted, but it is important to know that it is present. It is also important that the same filters used on the output signals be used on the input, or the system will try and match input frequencies to output frequencies that have been removed. Also noticed in the examination of the psds in Figure 7 was the relative lack of energy in the fourth floor signal at about 38 Hz. This could be due to a sensor malfunction, as all experiments showed a similar absence for the same sensor, and as there is no reason for the 38 Hz signal to not be present in only the fourth floor. However, the sensor may have been functioning correctly as the higher (~60 Hz) noise was recorded at about the same level as the other sensors. Just to be safe, in the identification process, the uncertainty for the fourth floor measurement was increased to the point where the algorithm ignored the updates from the fourth floor. This should not adversely affect the convergence of the algorithm, as it was shown (see memo "Extended Kalman Filter results for 10 DOF, 50 DOF, and NTS 5 DOF" of March 1, 2000), that the 10 DOF model would converge upon the correct answer easily with only 4 measurements. This, of course, is contingent upon the 10 DOF model describing the system with enough accuracy to allow convergence to occur.

In conclusion, there is fair agreement on the resonance locations using the MUSIC and PSD algorithms. They both indicate resonances at around 14.3, 32.2, and 37.8 Hz. The MUSIC algorithm seems to place the lowest resonance at only 2 Hz (and the first floor psd agrees) while the

other floor psds seem to indicate about 4.4 Hz. The fourth floor sensor data is probably unreliable due to the lack of frequencies around 38 Hz and it will not be used in the identification process.

2. Do the same thing to the innovation sequences. This will determine the location of the uncaptured modes of the system. Can also do it by comparing the spectra of the model and measured system.

Action: The psds of the recorded accelerations from STRUCOW5.MAT and the respective innovations are shown in Figure 8 for the 2000 points between 1 second and 6 seconds. It is clear that there are frequencies in the innovations that are not being modeled. In all cases, the recorded energy 14.4 Hz is not being modeled adequately. In floors 3 and 4, the peak at 37.6 is also not being modeled adequately, but it seems to be modeled correctly on the other floors. This makes it more difficult to identify as some floors are modeling the motion correctly while others are not.

We may also learn something from looking at the psds of the simulated floor signal using the recorded input signal. This will tell us where the model expects the resonances of the system to be. In figure 9, the psds of the recorded input signal and the simulated floor outputs are shown. The expected (see table 4, column 2) resonance at 5.2 Hz is easily identified except for the third floor, where it is conspicuously absent. This was also the floor where the force was applied, so it is likely that the response will be different there. The resonance at 13.2 Hz is also quite easy to see, and the peaks at 21.4 and 26.9 can just be made out. Thus, the model “expects” frequencies to be present near these resonances, and it is unknown why the model is unable to match the simulated peak at 13 Hz to the observed at 14.4 Hz.

3. Bandpass filter the data so that the modes the model is not able to capture are excluded from the data.

Action: From Section 1 and Figure 7, the recorded resonance locations are located at about 4.4, 14.4, 32.4, and 38.0 Hz for all channels except for the first floor, where the lowest resonance seems to be about 2.0 Hz. A 15th order bandpass (BP) filter was constructed using YULEWALK.M with passbands from 1.0-6 Hz, 12.5-16.5 Hz, 31-34 Hz, and 36.5-39.5 Hz. The frequency response of the BP filter is shown in Figure 10. It is not perfect, but it was the best I could do using a single filter stage with the parameters given. The FILTFILT.M command was used to minimize phase distortion and double the magnitude response of the filter. The recorded input and outputs from STRUCOW5.MAT were filtered and the resulting data used to identify the NTS structure. The identification program is EKF_NTS.M, and it uses fifteen parameters to attempt the identification. These parameters are the shear parameters α_y , stiffness parameters E, and length parameters L. Although the number of parameters (15) is larger than the number of DOF (5), this should not result in a problem. I have conducted tests using the 10 DOF simulated system and the k values of the K

matrix as my identification parameters, which far outnumber the 10 DOF. In these tests, convergence as observed only on those stiffness values which were independent of each other. The dependent stiffness values did not change. Thus in this case, if the identification parameters do not significantly affect the response of the structure, their values will remain unchanged. The results are not encouraging, and are summarized in Tables 2 and 3.

From Table 2 it is clear that the α_y parameters are the ones that were significantly changed, and therefore are the only ones that significantly affect the response of the system. Thus further studies will only use those parameters in order to speed up the processing time.

In Table 3 we see the results when only the α_y parameters are varied. The results are the same as for just about every other identification attempt: One of the parameters has grown to a large size, and the rest are all significantly decreased in size. It is not always the same parameter that grows, but it is almost always just one.

In Table 4 the resonance locations for the FEM, the EKF models (one with 15 identification parameters and one with 5), and the measured data are plotted. There is not a one-to-one correspondence between the locations as they do not place them all in the same frequency bands.

Using the 15 parameter EKF (2nd column), it seems that for 15 variables we have made a positive difference (11.1, 3.4, and 1.9%) in the estimation of the resonance locations observed at 4.5, 14.7, and 38.0 Hz. However, the 15 variable EKF still has pole locations at 0.7 and 9.3 Hz that are not observed, although the 0.7 resonance could be trying to model the observed 1.6 Hz.

Using only five variables, the improvement in estimation goes down for the observed locations at 4.5 Hz (only 4.4% better instead of 11.1) and 14.7 Hz (a disappointing 10.2% worse as opposed to 3.8% better). However, identification of the pole at 38.0 Hz has significantly improved from 1.9% better to 10.7% better, and there is also an indication that the resonance observed at 1.6 Hz in floors 1 and 2 (see Figure 7) is being modeled by the FEM.

In all of these tests $\dot{\mathbf{P}}$ was calculated every sample, as accuracy and not speed was more important in this calculation. Also, if a parameter was identified as negative, it was reassigned a value of (original estimate) / 10. This is because the quantities above cannot become negative and remain a physical model, and it was assumed that the real values should be within 90% of the estimate. There was no restriction placed on the size of the parameter if positive, so occasionally the change would be greater than 90%.

With the above restriction relaxed from 10% to 0.1%, the results were significantly worse, indicating that the 10% restriction is probably for the best. It might be better to restrict this even more in the future.

Returning to the default 10% restriction, I ran the ID algorithm with 30 seconds worth of data (12000 samples) instead of 5 seconds to see if that helped any. The results are shown in the fourth column of Table 5. It did not match the resonance at 4.5 Hz well, but matched the one at 14.7 Hz better than when 5 seconds of data were used. Still didn't converge, though.

Finally, I decided to use a series of high-performance Chebychev II notch filters to pass only the frequencies of interest to see if a better filter would help. The effect of five of these notch filters on a white noise input is shown in Figure 11. The top trace is the white noise spectrum, and then each one down from that is the spectrum of the noise after one of the notch filters is applied. The red trace on the last plot is the desired spectrum, and you can see that the performance is quite good.

So did this help? In a word, no. The fifth column of Table 5 shows the results of the experiment – the algo identified eigenfrequencies at 0.01, 3.7, 7.0, 9.4, and 15.1 Hz. The lowest resonance is too low, the 3.7 Hz is not a good approximation to the 4.5 Hz, the 7.0 and the 9.4 Hz are not observed in the data and there are no resonances associated with the strong peaks at 31.9 and 38.0 Hz. However, the resonance at 15.1 Hz is markedly (7.5%) better than the FEM, but this is the only bright spot in the picture.

Examining the PSDs of the innovations for this experiment (Figure 12) show that there is significant energy at 38.0 Hz for Floors 2-5 and at 14.7 Hz for Floors 4 and 5. This shows that for some reason the model cannot fit the resonance at 38.0 Hz. I tried filtering out first the 38.0 Hz band and re-doing the identification. The results are shown in the sixth column of Table 5, and although the 4.5 Hz peak is modeled quite well the rest of the resonance locations are questionable. Lastly, I also removed the 14.7 Hz band (seventh column) and repeated the identification, with even poorer results. Part of the problem here is that the innovations in Figure 12 do not have the same energy at the same locations for all the floors. Perhaps it would be better to filter only the floors with nonwhite innovation energy, i.e. for 14.7 Hz we would only filter the 4th and 5th floors. I don't think it will make that much difference, but it may be worth a try.

Basically, all of these models suffered from the same problem – the solution never really converged. Almost all the parameters would get driven down as low as possible (to about 10% of their starting value) with perhaps one or two at about the same magnitude. The resulting simulated accelerations were very large, on the order of 10x what was measured, leading to large innovations and poor overall fits. I believe that this model is simply inadequate to describe the dynamics of the NTS structure.

4. Use a simple grid-search or similar method to calculate the initial estimates of the stiffness parameters. Then run the EKF with the new estimates.

Action: Unfortunately, I did not have time to implement this.

I think that's about it. If I have missed anything or if you have any further suggestions let me know.
Thanks again for coming!

Greg

Tables

Exp #	Threshold (%) or parts calculated	Run time* (min)	Samples to converge	ODE calc skipped (%)	Mean ID error (%)	Max ID error (%)
1	None	5.88	127	0	0.62	1.68
2	None	6.59	127	0	0.62	1.68
3	10^{-16}	7.99	127	0.002	0.63	1.68
4	0.001	7.93	127	0.17	0.66	1.79
5	0.01	7.88	127	0.39	0.65	1.78
6	0.1	8.15	127	1.03	0.38	0.79
7	0.1	7.67	127	1.03	0.62	1.61
8	1.0	8.24	134	4.65	0.30	0.65
9	1.0	6.95	134	4.84	0.35	0.77
10	2.5	19.48	N/A	8.18		
11	5.0	18.54	N/A	15.2		
12	Only stiffness	8.99	N/A	95.4	30.8 (8.20) ¹	77.6
13	Every 5 th with stiffness	10.0	~150	75.4	1.16 (1.39) ¹	-2.84
14	Every 10 th with stiffness	9.96	~150	84.2	3.35 (1.97) ¹	-7.88
15	Every 5 th w/o stiffness	9.62	~100	50.0	0.95 (1.65) ¹	0.95
16	Every 10 th w/o stiffness	9.77	~150	90.0	1.97 (1.65) ¹	-3.98

Table 1. Convergence times and identification errors for the same input/output sequences and different ΔP thresholds and conditions. Every n^{th} with stiffness means that the Pdot values associated with the stiffness states were calculated at each time sample and the entire Pdot calculation was done every n samples. Every n^{th} without stiffness means that only the entire P calculation was done every n samples.

* run time can vary depending on the amount of memory available, accurate to about ± 1 minute

¹ average error using stiffness values defined by the mean of the last two hundred estimates

Identification parameter	Model value	Estimated value	% difference
α_y 1	0.00216	0.0000156	99.28
α_y 2	0.00216	0.000528	75.575
α_y 3	0.00216	0.000216	90.0
α_y 4	0.0018	0.000341	81.06
α_y 5	0.0018	0.005210	-189.44
E_1	1.06 e7	1.06 e7	0
E_2	1.06 e7	1.06 e7	0
E_3	1.06 e7	1.06 e7	0
E_4	1.06 e7	1.06 e7	0
E_5	1.06 e7	1.06 e7	0
L_1	25.5	25.5	0.00081
L_2	25.0	24.998	0.00705
L_3	25.5	25.5	-0.00028
L_4	22.0	22.0	-0.00011
L_5	26.5	26.5	-0.00178

Table 2. Results of NTS 5 DOF identification where a simple filter was used to try and remove the modes that were not being fully addressed by the model. Here 15 variables are used to do the identification. It is clear that the α_y coefficients are the only ones that affect the response of the model, in the future they are the only ones used.

Identification parameter	Model value	Estimated value	% difference
α_y 1	0.00216	0.009227	-327.16
α_y 2	0.00216	0.000216	90
α_y 3	0.00216	0.000345	84.015
α_y 4	0.0018	0.00018	90
α_y 5	0.0018	0.00018	90

Table 3. Results of NTS 5 DOF identification where a simple filter was used to try and remove the modes that were not being fully addressed by the model. Only the α_y parameters were varied in an attempt to identify the system. Three of the parameters have been reduced to their minimum allowed values (denoted by a 90% change), indicating that convergence has probably not been attained.

Recorded resonance locations (Hz)	FEM e-frequencies in Hz (error %)	EKF 15 variable calc e-freq in Hz, (improvement %, + good)	EKF α_y variable calc e-freq in Hz, (improvement %, + good)
1.6		0.7	2.1
4.5	5.2 (-15.5)	4.3 (11.1)	5.0 (4.4)
		9.4	7.6
14.4	13.2 (8.3)	13.7 (3.4)	11.7 (-10.5)
	21.4		
	26.9		
31.9	33.6 # (-5.3)	34.3 (-2.2)	
38.0	33.6 # (9.7)	34.3 (1.9)	37.7 (10.7)

Table 4. Comparison between the original FEM e-frequencies, the EKF e-frequencies (once using all 15 variables to identify the structure, and once using only the α_y values), and the location of the resonances of the recorded measurements' spectrum from STRUCOW5.MAT. Neither model has two peaks above 30 Hz, and it is unclear which recorded resonance they are attempting to match. All were using 5 seconds of data, the noise parameters were $\Delta Q_x = \Delta P_x = 0.02$, $\Delta Q_v = \Delta P_v = 0.4$, $\Delta Q_k = \Delta P_k = \text{pars1}$, $\Delta R_m = 0.2$, $\Delta R_4 = 1e^{12}$.

= not sure if the algorithm fitting the recorded peak at 31.9 or 38.0 Hz

Recorded resonance locations (Hz)	FEM e-frequencies in Hz, (error %)	EKF with simple BP 5 sec data (improvement %, + good)	EKF with simple BP 30 sec data (improvement %)	EKF with notch filters (improvement %)	EKF with notch filters, 38 Hz block filtered out	EKF with notch filters, 14.7 and 38 Hz blocks filtered out
1.6		2.1	1.2, 3.0	0.01	1.4	0.4
4.5	5.2 (-15.6)	5.0 (4.5)	5.3 (-2.2)	3.7 (-2.2)	4.5 (15.6)	2.3 (-33.3)
		7.6	8.3	7.0, 9.4	6.9, 9.6	8.4, 8.5
14.7	13.2 (10.2)	11.7 (-10.2)	13.4 (0.5)	15.1 (7.5)	N/A	Filtered out
	21.4					
	26.9				107.2	
31.9	33.6 (-5.3) #					30.3 (0.3)
38.0	33.6 (11.6) #	37.7 (10.8)	N/A	N/A	Filtered out	Filtered out

Table 5. Eigenfrequency locations for the 5 DOF NTS model using two different filters (simple BP and 5 stage notch) to limit the frequencies available to the identification algorithm. The first column is the observed locations, the second is the FEM locations (with error in %), and the rest are the different filters (with improvements over the FEM locations in parenthesis). All except column 4 were using 5 seconds of data, the noise parameters were $\Delta Q_x = \Delta P_x = 0.02$, $\Delta Q_v = \Delta P_v = 0.4$, $\Delta Q_k = \Delta P_k = \text{pars1}$, $\Delta R_m = 0.2$, $\Delta R_4 = 1e^{12}$.

Figures

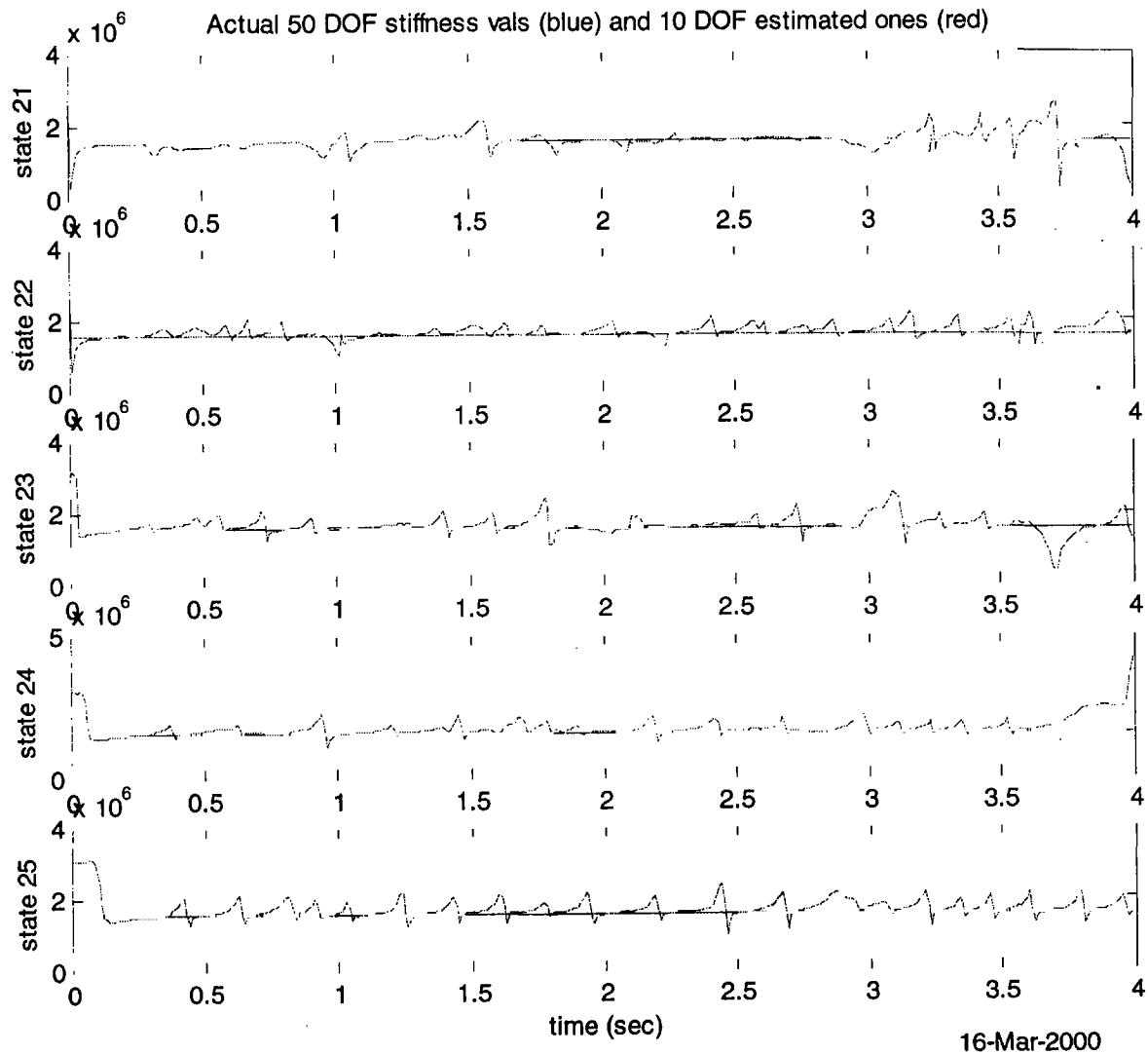


Figure 1. The identification estimates using the 10 DOF model and the 50 DOF simulated system outputs. 1000 samples were used, the first 400 are shown. All initial estimates were $3.1e^6$, all actual values were $1.55e^6$. The large divergence of state 24 near 4 seconds was corrected by 4.2 seconds.

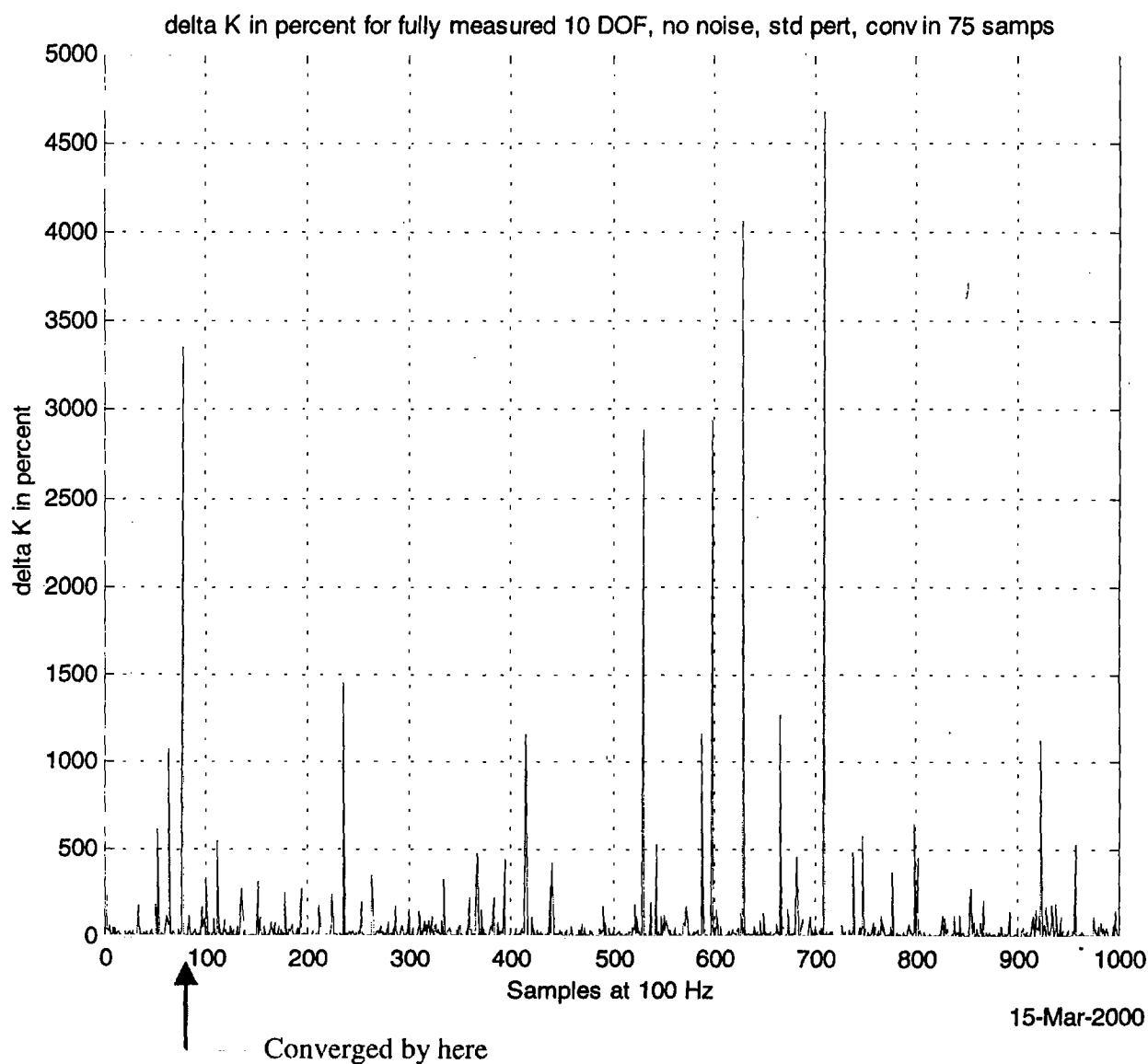


Figure 2. The change in K (in percent) for the 10 DOF system out to 1000 samples at 100 samples/second. The algorithm had converged to within 1% error by 75 samples.

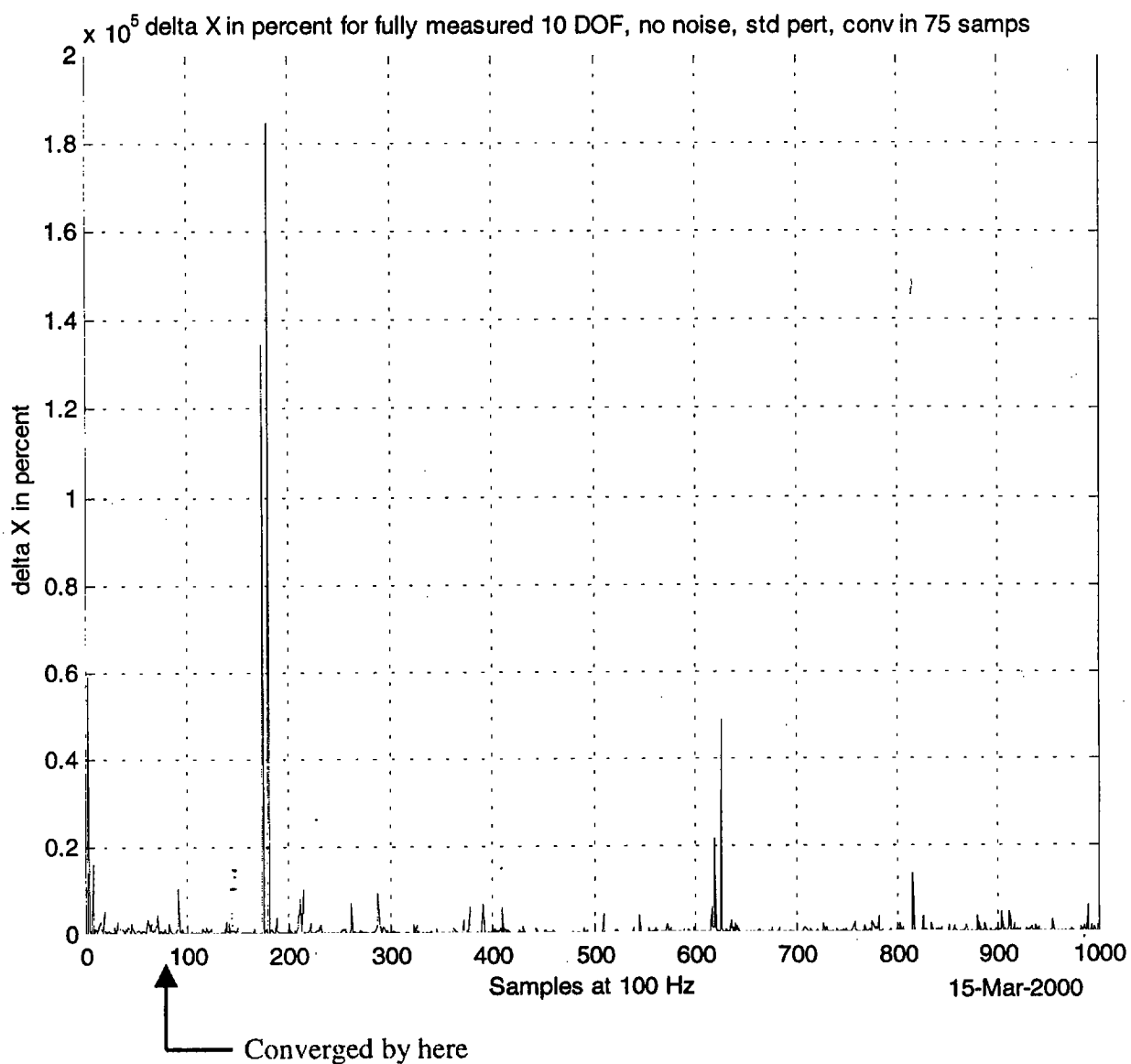


Figure 3. The change in all states X in percent for 1000 samples at 100 samples/second. Note the scale here is multiplied by 10^5 , the maximum value being $2 \times 10^5\%$.

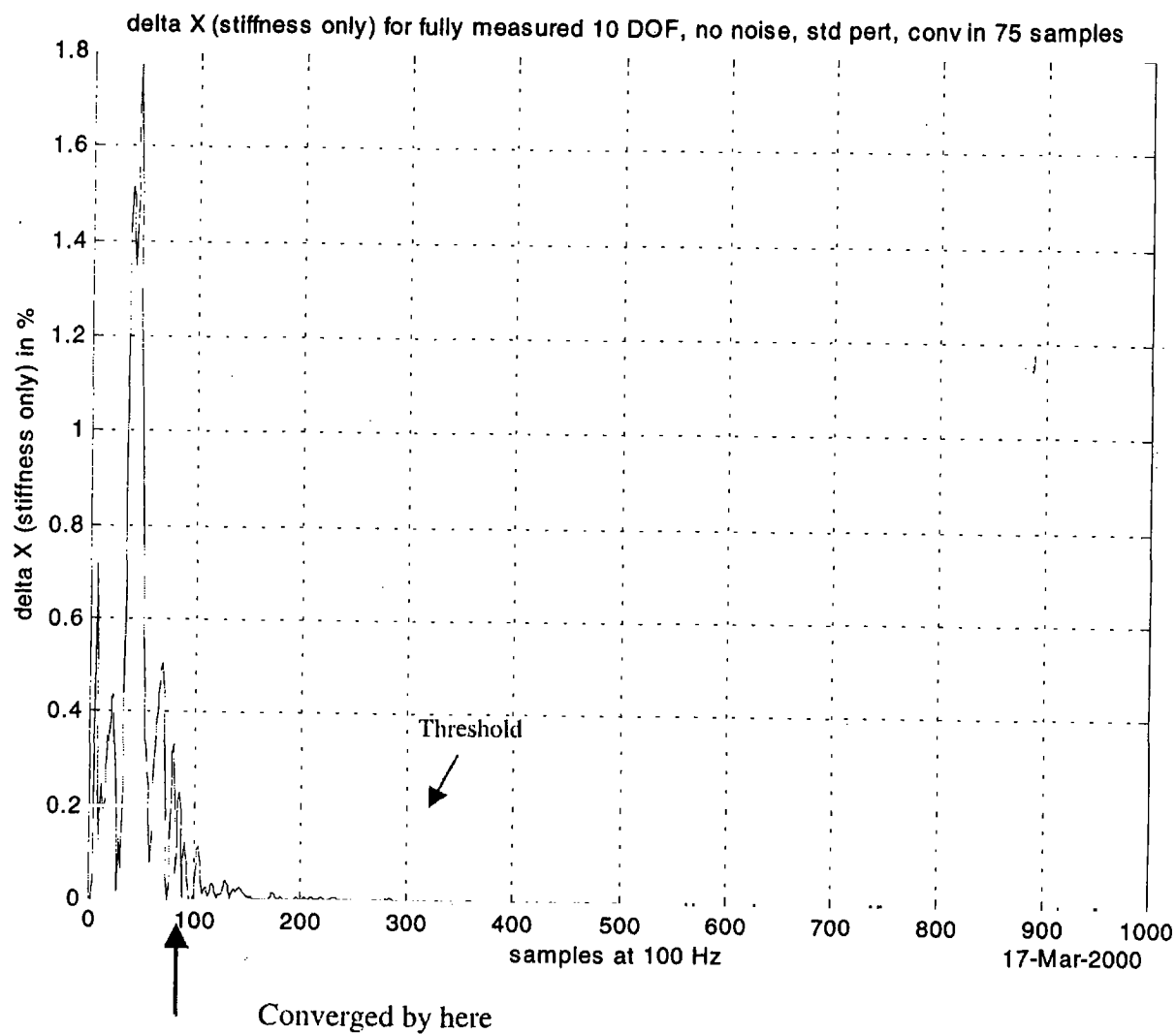


Figure 4. The change in only the stiffness states in percent for 1000 samples at 100 samples/second. Note the scale here is much smaller, the maximum value being only 1.8%.

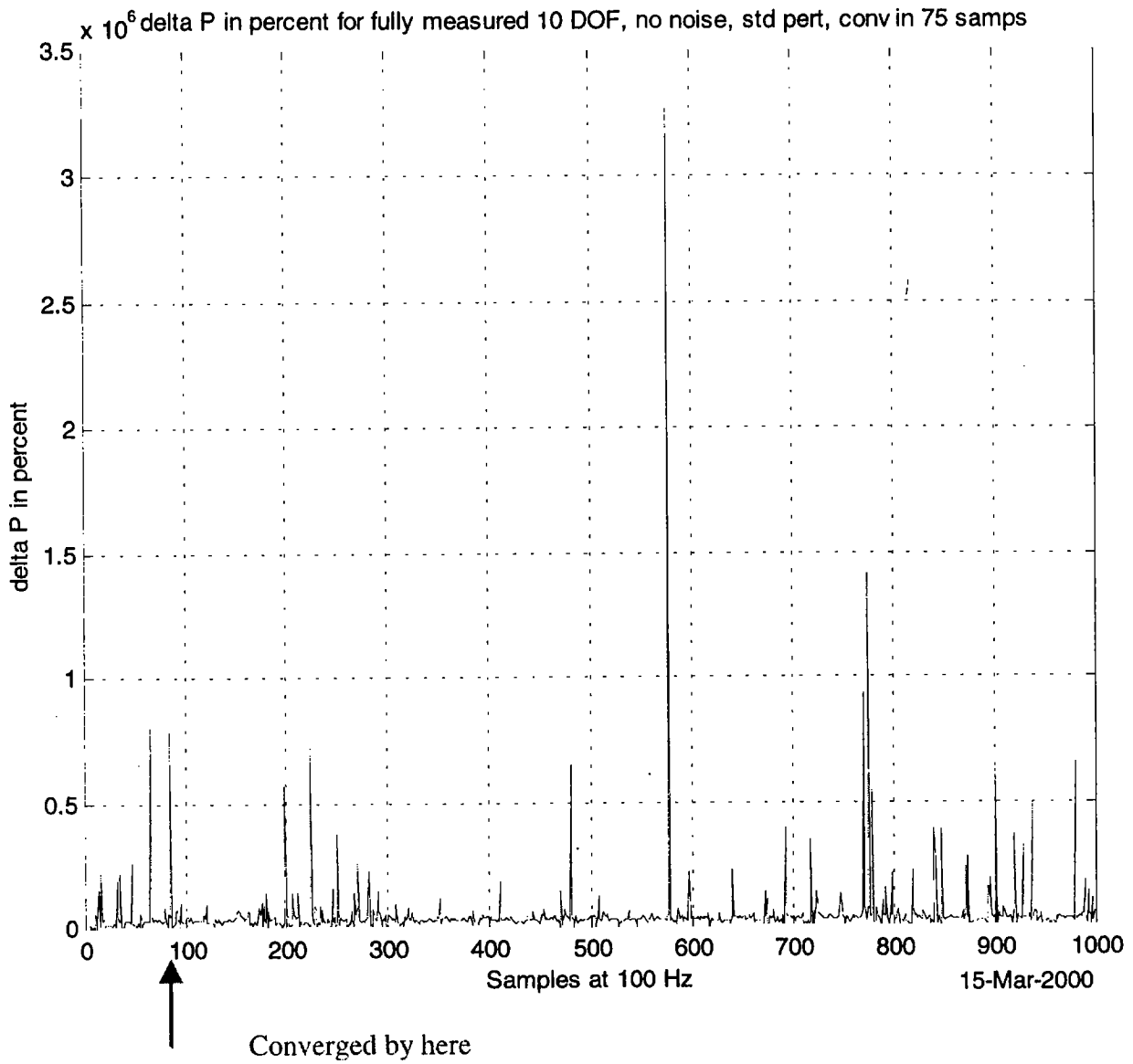


Figure 5. The change in error covariance P in percent for 1000 samples at 100 samples/second. Note the scale here is multiplied by 10^6 , the maximum value being $3.5 \times 10^6\%$.

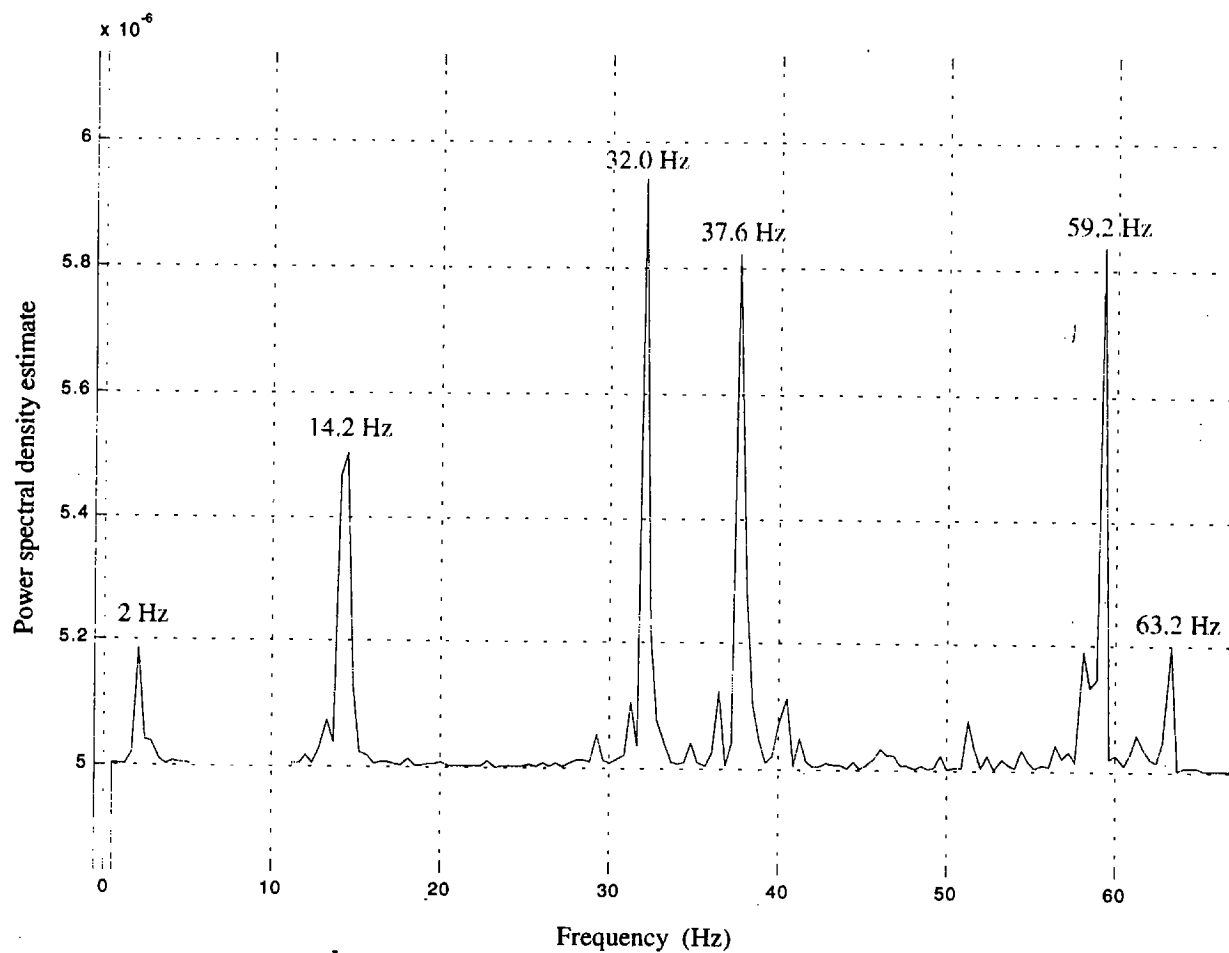


Figure 6. MUSIC spectrum of five +y direction outputs from strucow5.mat, first 5 seconds, 1000 points (pmusic.m).

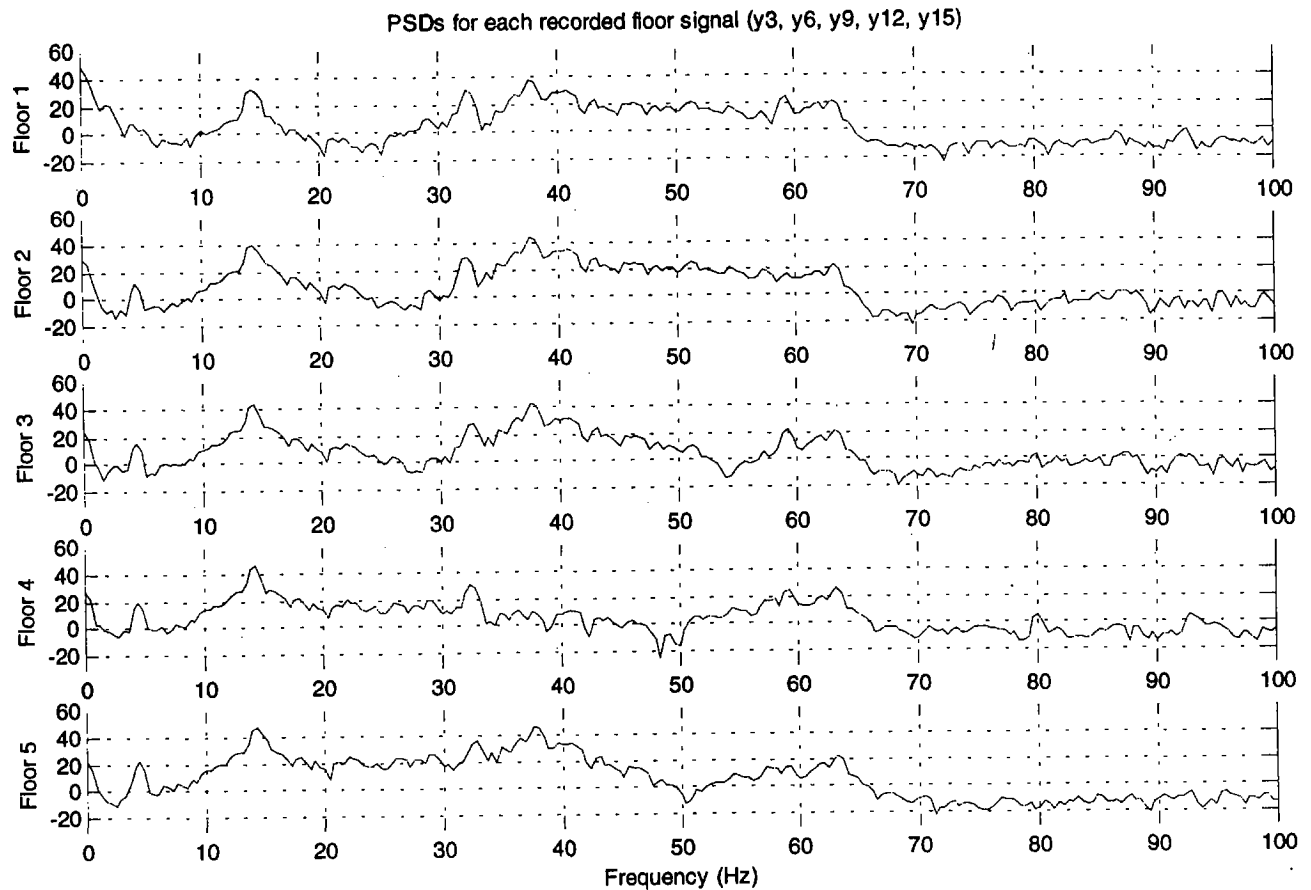
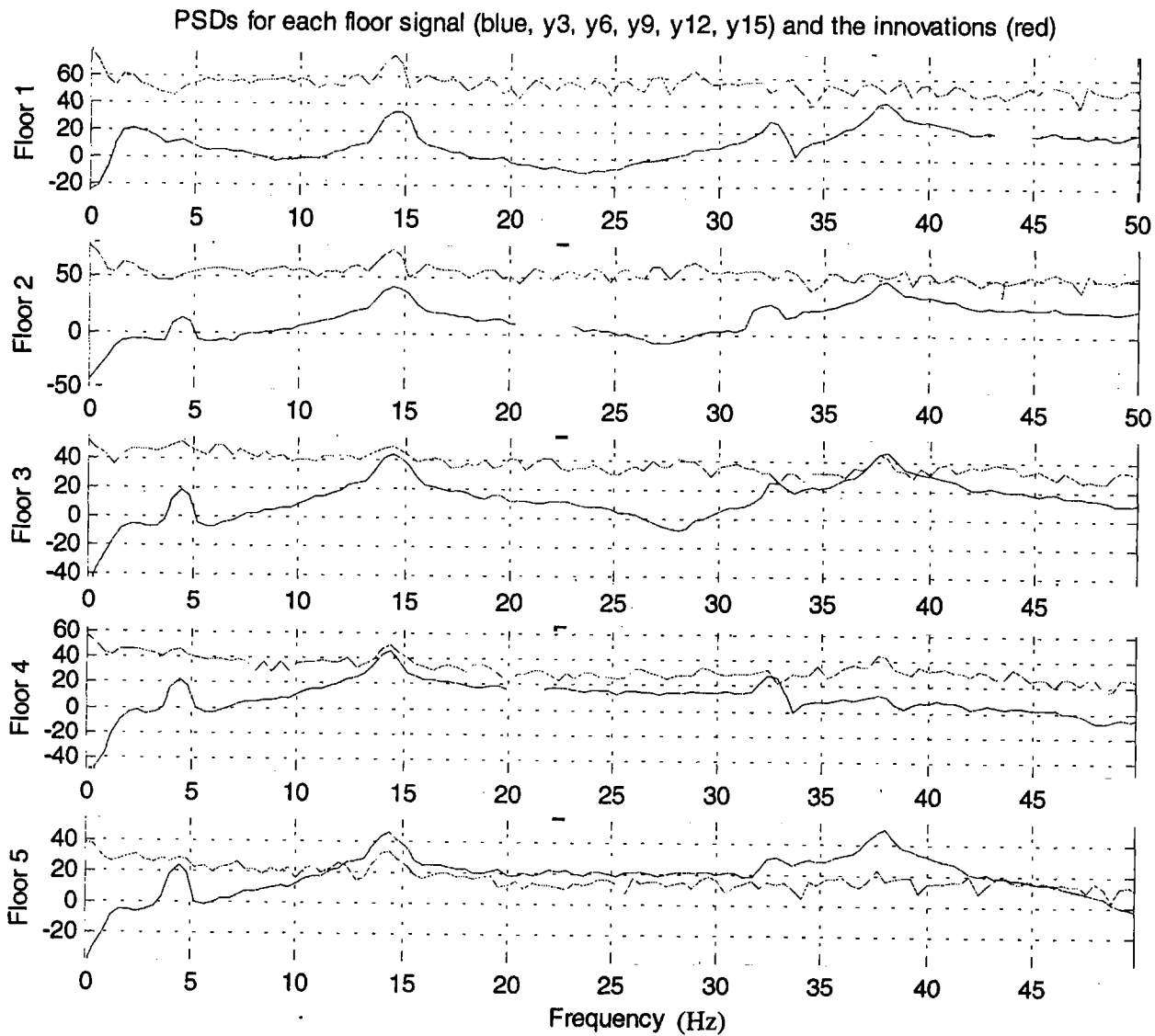


Figure 7. The power spectral density estimates (psds) of the recorded y-direction outputs (y3, y6, y9, y12, and y15) for the first 30 seconds of strucow5.mat. The major peaks are at approximately 4.4, 14.4, 32.4, 38.0, 59.5, and 63.2 Hz. There is also significant energy at about 1.5 Hz for the first floor (top plot), and a significant lack of energy at 38 Hz for Floor 4.



22-Mar-2000

Figure 8. Power spectral densities (psds) of the recorded data from the five floor sensors monitoring acceleration in the +y direction (blue, lower trace) and their innovations (red, top trace). Any nonwhite energy in the innovations means that the model is not sufficiently describing the system near that frequency. The data have been HP filtered above 1 Hz to remove DC drift.

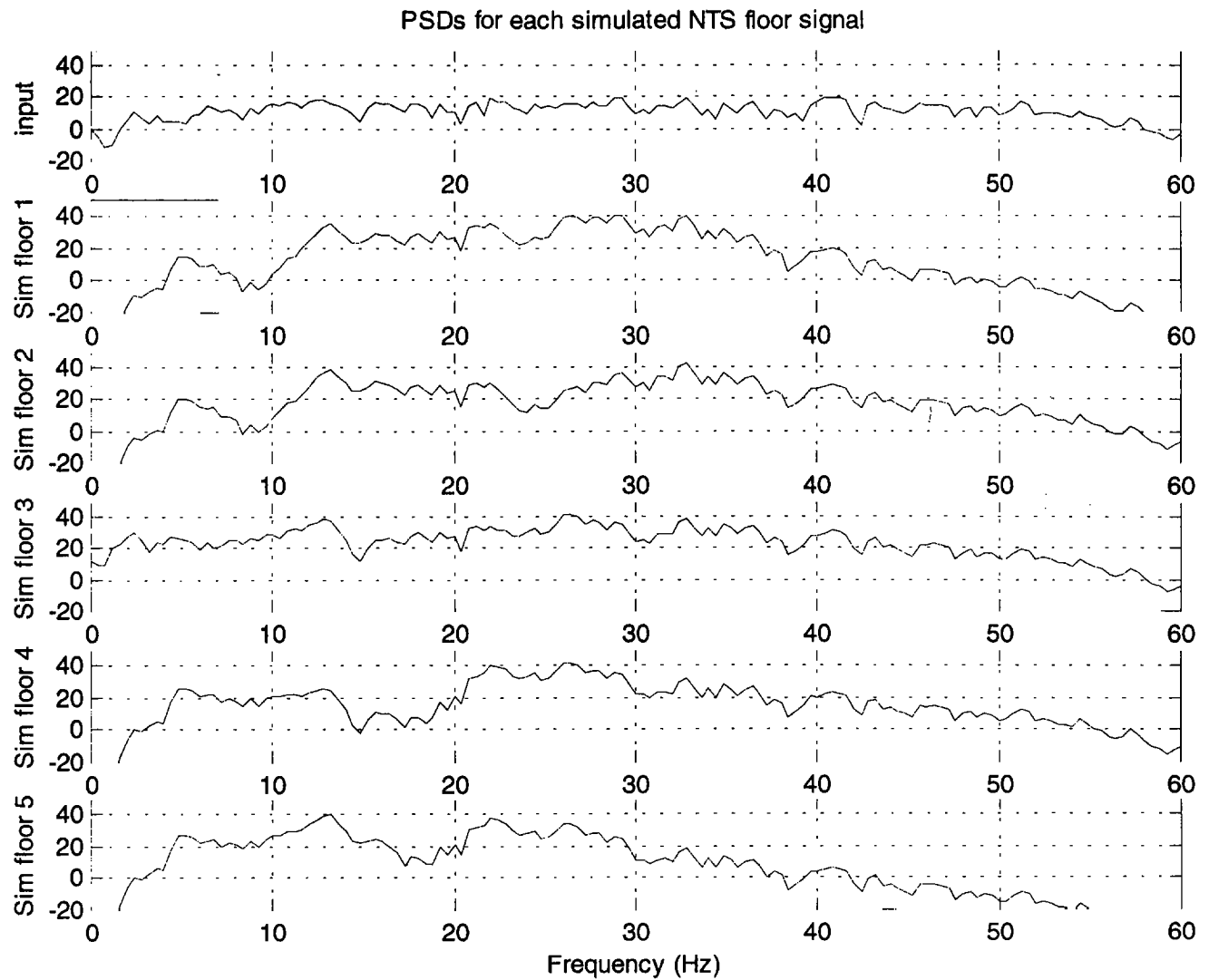


Figure 9. PSDs of the simulated output from the NTS 5 DOF model using the recorded input from strucow5.mat. The only two recognizable resonances are at about 5.0 and 13.0 Hz. Note the relative lack of structure compared to Figure 7.

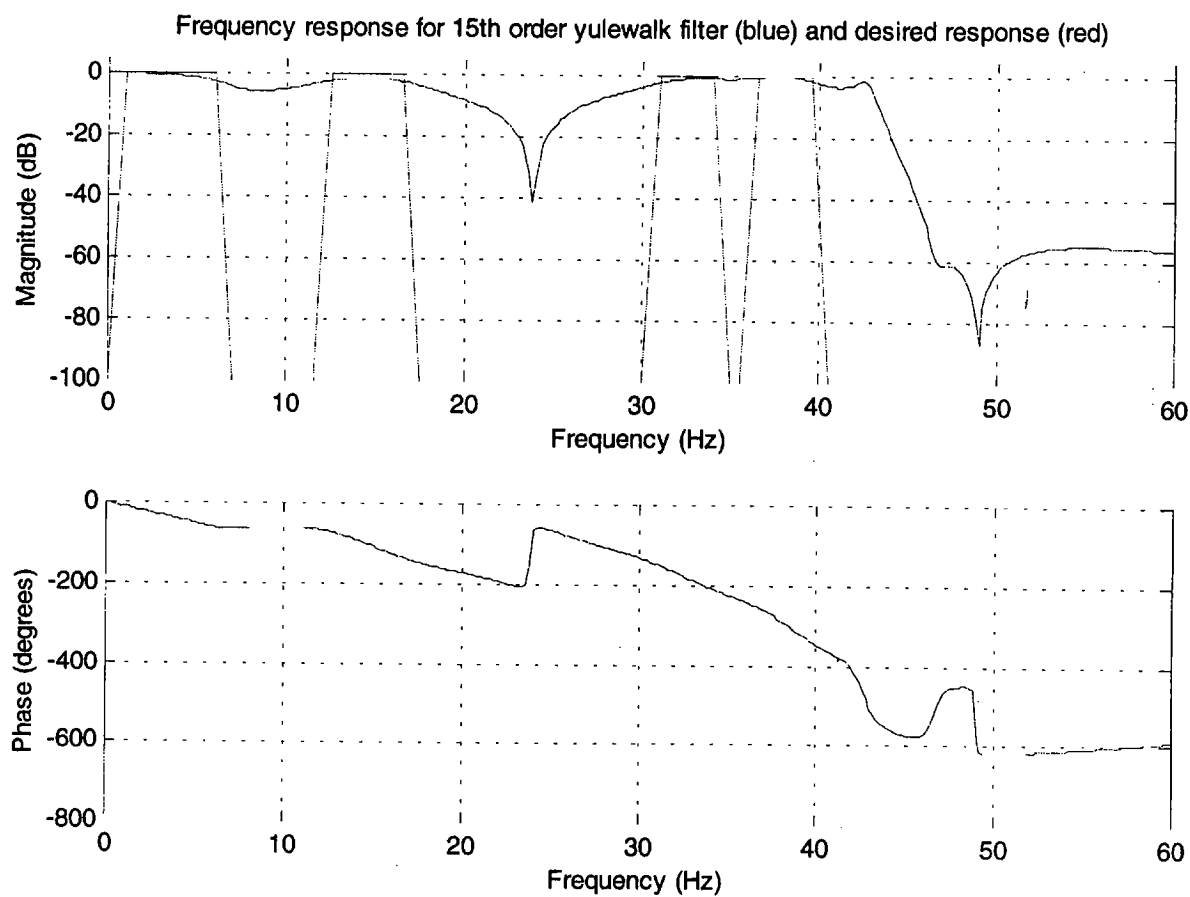


Figure 10. Frequency response for single stage multiple bandpass filter (blue) and the desired response (red). Note the poor fit when only one stage is used.

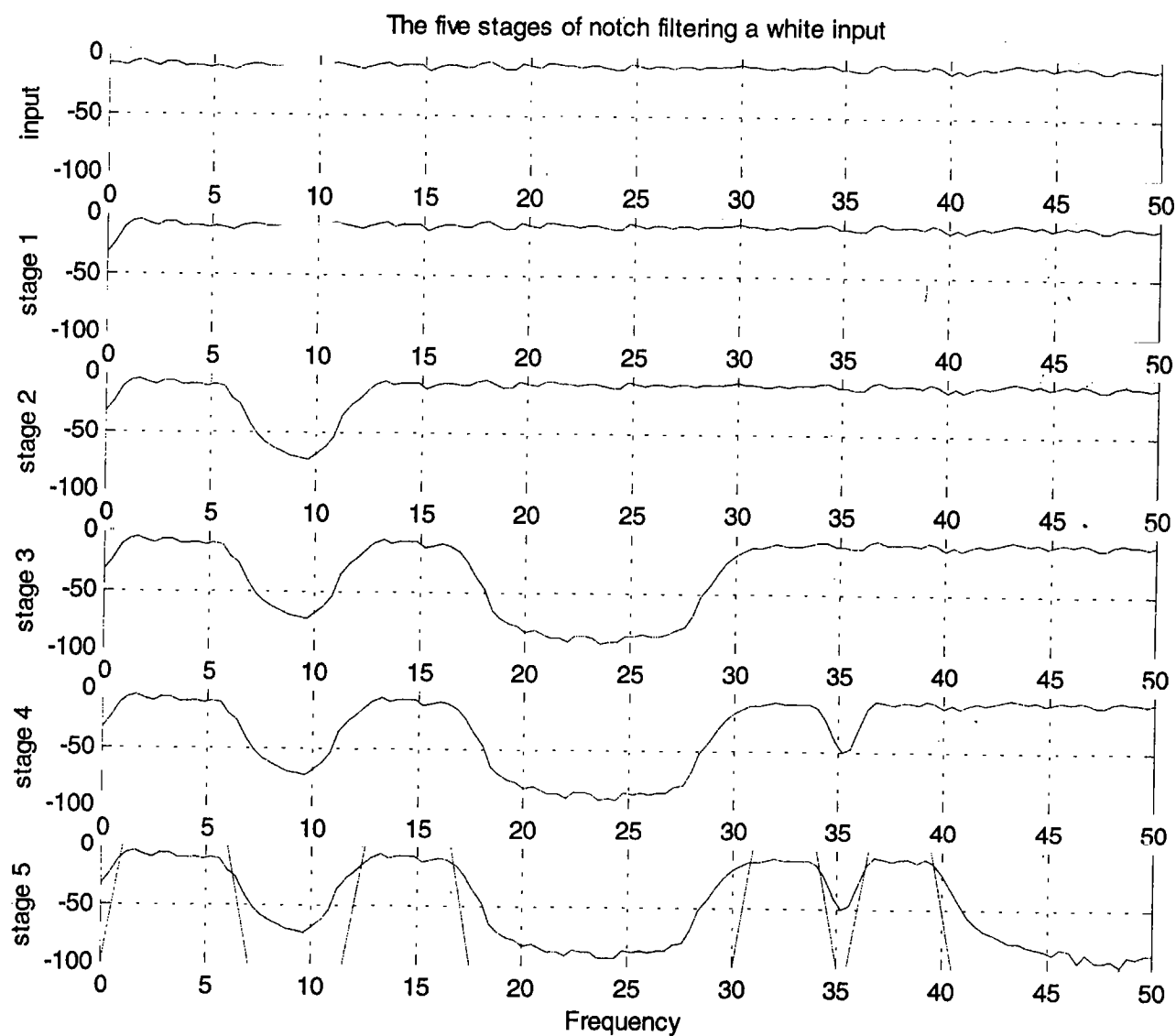


Figure 11. The five stages of notch filtering a white noise input so that information is only passed in the frequency bands of interest. Five high performance Cbebychev II notch filters were used to achieve the desired spectrum (red trace on lowest plot).

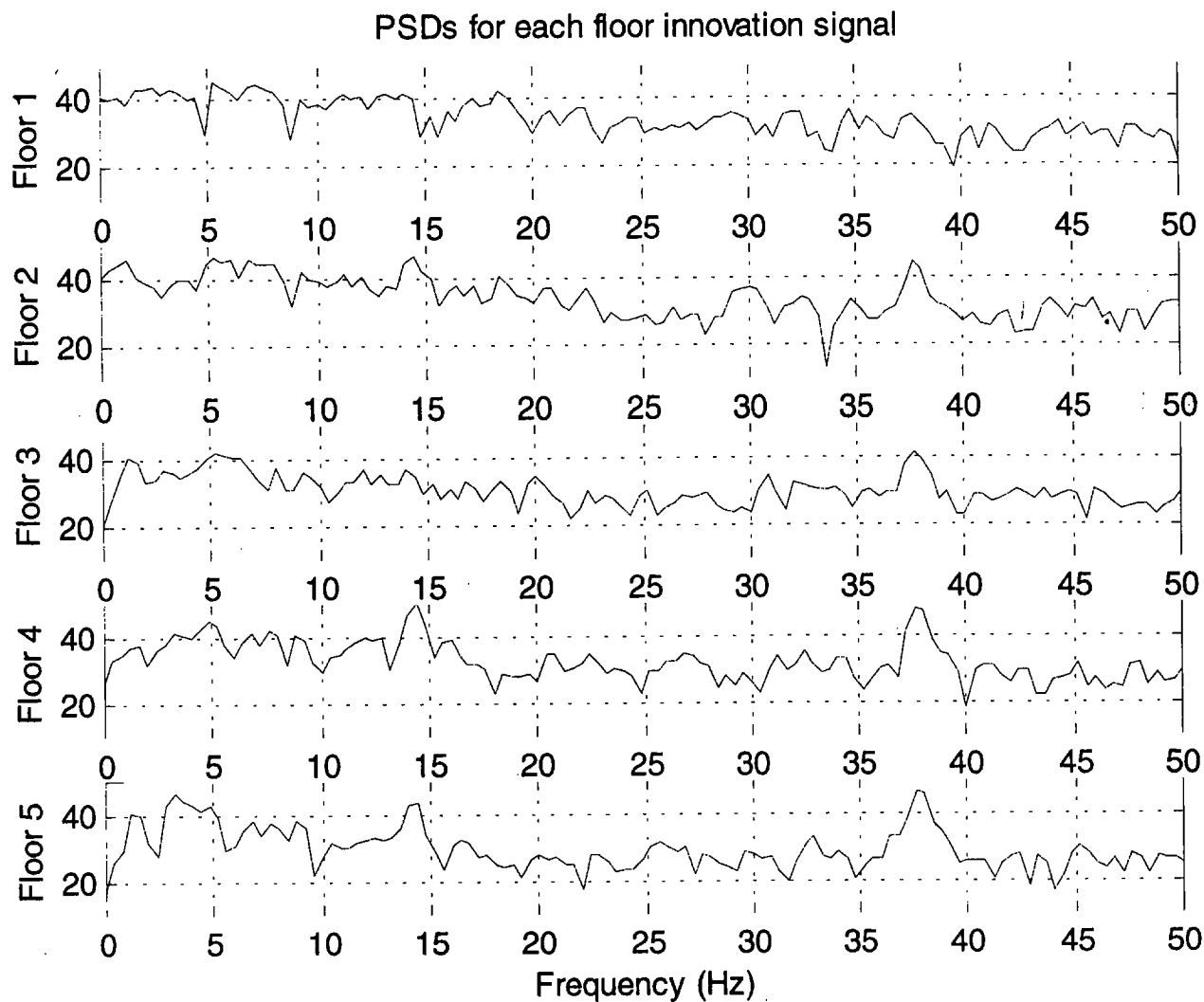


Figure 12. PSDs of the innovations while using the notch filter of Figure 11. The peak at 38.0 Hz and the smaller one at 14.7 Hz indicate the model is not describing the system well near those frequencies.

